UNIVERSIDAD CARLOS III DE MADRID

DEPARTAMENTO DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Desarrollo de un "escaparate caliente" con Kinect

Una prueba de concepto

Autor: Jose Enrique Sibón Sancho

Tutor: Miguel Ángel Patricio Guisado

Colmenarejo, 25 de junio de 2013

Ante todo, con calma.
¡Si es que siempre lo he dicho!

Original

You can make sense out of this? Do tell! I've been staring at it for the last 20 minutes trying to find some modicum of an English word among the mountains of what basically amounts to this guy eating his keyboard!

A friend of a friend, in his signature

Agradecimientos

A mis padres, por tolerar mis horarios extremos y por estar siempre dándome consejos útiles sobre el trabajo en concreto y la vida en general. Pero especialmente a mi madre, por animarme siempre a que me pusiera a escribir el presente documento con sus "Deja ya de jugar a las máquinas y ponte con la memoria", aunque ya estuviese con ella. A mi hermano, por estar siempre dispuesto a explicarme lo que él sabe y yo desconozco sobre la anatomía y la medicina, un gran mundo de preguntas e inspiración para desarrollar IAs.

A mis antiguos amigos del colegio, incluso a los que he perdido, que me habéis guiado y moldeado para ser lo que soy. Siempre os estaré agradecido.

A mis amigos de la Universidad, que en estos cinco años hemos pasado de todo juntos, como una gran piña. Carlos, Sergio, Nico, Iñaki... tened por seguro que estaré aquí siempre para lo que necesitéis, podéis contar con ello.

A mi tutor actual, Miguel Ángel Patricio y a mi tutor anterior, Antonio Berlanga, a los cuales he traído de cabeza con mis escasos correos y frecuentes desapariciones. Por darme siempre buenas ideas sobre cómo y a dónde dirigir el desarrollo del proyecto.

Finalmente, a mis amigos de internet. Tanto si es para pasar un buen rato como si es para tener una conversación profunda, siempre estáis ahí. Por mucho que os haga esperar.

Índice

Índice de figuras

Índice de tablas

Introducción

Desde que salió al mercado, el dispositivo Kinect supuso una revolución en muchos aspectos. No sólo los diseñadores de videojuegos advirtieron su potencial, sino también muchos científicos, empresarios e informáticos por igual. A estas alturas, ya se han desarrollado una miríada de aplicaciones que hacen uso de este dispositivo de una u otra forma, pero el potencial de Kinect para otras nuevas sigue siendo enorme.

En cierto modo, es emocionante que se pueda contribuir a buscar aplicaciones de Kinect que puedan ayudar en el mundo real. La tecnología que se condensa en este dispositivo es tanta y tan asequible comparada con las alternativas que se usaban hasta ahora que es lógico pensar que la primera aplicación real que se verá de Kinect, aparte del ámbito de los videojuegos, será en el entorno comercial.

Siguiendo esta línea de pensamiento, se creyó que sería interesante realizar una prueba de concepto de una posible aplicación real, para comprobar la plausibilidad del uso del dispositivo Kinect en los comercios. Las restricciones del dispositivo son numerosas, pero una vez se adapte el entorno correctamente para el funcionamiento óptimo de Kinect, las ventajas que se obtengan de su uso pueden ser muchas.

Ambientación y motivación

El concepto del que se pretende probar la viabilidad es la integración del dispositivo Kinect en un escaparate. En el entorno comercial, el escaparate es una pieza clave, y saber si la disposición del mismo es buena podría ser vital para un comercio. En este sentido, el ámbito a tratar es la publicidad, o más bien, las reacciones que produce la publicidad en los observadores.

Tras el cristal de un escaparate, uno siempre puede ver una serie de artículos dispuestos de forma artística, o quizá caótica, a lo largo del espacio que hay para mostrarlos. Sin embargo, ¿están dispuestos de la forma adecuada? ¿El escaparate es suficientemente atrayente? Hasta ahora, estas y otras preguntas relacionadas con el escaparate se podían comprobar aproximadamente. Si entran muchos clientes, la distribución del escaparate es buena, y si entran pocos, probablemente sea mala.

El diseño de escaparates se ha convertido prácticamente en un arte, y no son pocas las empresas que ofrecen servicios relacionados con este tema. Por un precio, un escaparatista acude al comercio y dispone el escaparate para que llame la atención de los potenciales clientes y los incite a entrar. También están los servicios que simplemente indican cómo organizarlo y son los empleados del comercio los que realizan los cambios oportunos.

Sin embargo, la efectividad de estos diseños está por comprobar. Es cierto que se basan en estudios y en la psicología de las personas, pero un comercio no posee datos reales de la mejora que ha supuesto un diseño concreto en su caso particular. Este problema es el que se puede atajar por medio de la implantación de un dispositivo Kinect en el escaparate.

La cámara de Kinect puede observar a los que miran el escaparate, calcular a qué lugar están dirigidas sus miradas y discernir si un artículo les llama la atención o no. Con este sistema, el comercio tendría a su disposición datos reales de qué es lo que llama más la atención y qué es lo que sobra en ese escaparate. Una información que podría resultar muy útil. Además, dado que Kinect posee varias cámaras, también se puede contemplar la posibilidad de obtener información adicional de la persona.

Es por eso que se pretende realizar una prueba de concepto del sistema, para comprobar la validez de un sistema con una clara aplicación y utilidad, y para trabajar con el dispositivo Kinect, una herramienta con un gran potencial. Aunque al final los datos indicasen que el sistema no tiene futuro, habría merecido la pena el tiempo invertido en ello, puesto que gran parte de lo documentado o implementado durante el proceso podría ser reutilizado más adelante.

Objetivos

Los objetivos del presente proyecto de fin de grado son varios. En primer lugar, se trata de explorar las funcionalidades del nuevo SDK de Kinect para Windows y poder experimentar con el servicio de reconocimiento del esqueleto que éste proporciona. Siempre se tuvo curiosidad por cómo funcionaba, y a la elección de este proyecto en concreto tuvo gran influencia el hecho de que se iba a utilizar el dispositivo Kinect.

En segundo lugar, es también un objetivo del presente proyecto el explorar el servicio de reconocimiento facial que también ofrece el SDK de Kinect, puesto que es un servicio muy interesante que puede tener diversas aplicaciones.

La presencia de tanto una cámara en color como de una cámara de profundidad en el dispositivo Kinect da pie a que se puedan obtener varios tipos de datos sobre las personas que capten dichas cámaras. Por esa razón, otro objetivo de este proyecto es el de explorar los tipos de datos que se pueden obtener al combinar la información captada por ambas cámaras; rango de edad, sexo, talla de ropa, etc.

El último objetivo de este proyecto consiste en la realización de una prueba de concepto que demuestre la viabilidad de integrar el dispositivo Kinect en el área de la publicidad del entorno comercial.

Para ello, se desarrollará una aplicación parcial que se ejecutará en un dispositivo Kinect instalado en el escaparate de un comercio. Este software permitirá calcular a dónde está mirando un potencial cliente y cuánto tiempo dedica a cada objeto del escaparate. Se pretende, además, que el software obtenga información del potencial cliente y la muestre junto con el resto de datos obtenidos.

Así pues, se planea implementar un software para el dispositivo Kinect que sea capaz de:

- Obtener la dirección de mirada de una persona.
- Calcular a partir de ella la posición del escaparate que el sujeto está mirando.
- Discernir cuáles de los objetos a los que ha mirado le han llamado la atención al sujeto.
- Obtener algún tipo de información de la persona que está mirando el escaparate. Se busca que use alguna de las formas de aprendizaje automático, pero dependerá de lo que se obtenga al explorar la capacidad de obtener datos de los sujetos.
- Mostrar los datos obtenidos sobre los que miraron al escaparate.

Terminología

En este apartado se describen algunos de los términos empleados en este documento:

- **API:** Abreviatura formada por las siglas de "Application Programming Interface", Interfaz de Programación de Aplicaciones en inglés. Es un conjunto de métodos y clases que se pueden usar desde una aplicación para que el código nativo de otra aplicación o pieza de hardware o software realice ciertas acciones. A *grosso modo*, es la forma que tiene un programa de interactuar con otro anterior sin tener acceso a sus datos internos.
- **Entorno de programación:** Se trata de un software que permite al programador controlar y realizar todo lo relacionado con el proceso de programación. Entre otras tareas, permite organizar el código fuente en paquetes, añadir y manejar referencias a recursos externos, compilar un código en un ejecutable y realizar tareas de debug (inspección de la ejecución línea a línea) en el código fuente.
  Los más modernos tienen funcionalidades adicionales para comodidad del programador, como autocompletar nombres de métodos y variables, permitir buscar las definiciones y referencias de los elementos del código, cambiar el nombre de una variable a la vez en todos los lugares donde se usa o realizar precompilaciones para indicar algunos errores de programación mientras se escribe.
- **RGB:** Abreviatura formada por las siglas de "Red, Green, Blue", Rojo, Verde, Azul en español. Es uno de los sistemas existentes de codificación del color, por lo que se usa en una gran cantidad de dispositivo que

obtienen o usan imágenes en color. El sistema se basa en representar cualquier color como una combinación de esos tres, lo cual es posible ya que son los tres colores básicos de la luz.

- **SDK:** Abreviatura formada por las siglas de "Software Development Kit", Kit de Desarrollo de Software en inglés. Se trata de un conjunto de herramientas de desarrollo que se proporcionan para permitir a un programador construir aplicaciones para un sistema o plataforma concreto. Puede incluir desde un API básico para permitir el uso de un cierto lenguaje de programación hasta piezas hardware que se requieran para tareas específicas.
- **Tubería:** Una tubería consiste en una cadena de procesos en la que la salida de uno supone la entrada para otro. Se suelen utilizar búferes de datos intermedios para evitar esperas.

Medios

Para la realización del presente proyecto se requiere una serie de medios, tanto físicos como de software, que se detallan a continuación.

Ordenador

Prácticamente todas las labores que se requieren para llevar a cabo este proyecto se pueden realizar en un ordenador. Se tiene acceso a dos ordenadores por la duración del proyecto, uno es una estación de trabajo cedida por la Universidad, mientras que otro es un ordenador portátil personal. Ambos tienen instalados el mismo entorno de programación y la última versión del SDK de Kinect, que se describen más adelante en esta misma sección.

Estación de trabajo

La estación de trabajo es un ordenador de sobremesa perteneciente a la Universidad Carlos III de Madrid. Se ha cedido el uso del mismo para poder realizar el desarrollo del proyecto en el laboratorio de informática denominado GIAA (Grupo de Inteligencia Artificial Aplicada) y tener todos los medios necesarios al alcance de la mano.

Las características de la estación de trabajo son las siguientes:

- Procesador Intel Core i5-2300 de 4 núcleos a 2.80 GHz.
- 4 GiB de memoria RAM.
- Tarjeta gráfica integrada.

Portátil

El ordenador portátil personal se usará fundamentalmente para las pruebas de validación y para la redacción de la presente memoria, ya que son tareas que no es necesario realizar en el laboratorio de informática.

Figura 1: Aspecto externo del dispositivo Kinect.

Las características del ordenador portátil son:

- Procesador Intel Core i7-2630QM de 4 núcleos multiproceso a 2 GHz.
- 6 GiB de memoria RAM.
- Tarjeta gráfica NVIDIA GForce GT 540M.

Kinect

El dispositivo Kinect, también llamado sensor Kinect o Kinect a secas, es una pieza de hardware que contiene una cámara de color, una cámara de infrarrojos, un conjunto de micrófonos y un acelerómetro. Además, la base está motorizada para poder inclinar el dispositivo usando comandos desde el software, lo que aumenta su utilidad. Físicamente, tiene el aspecto que se puede apreciar en la Figura 1.

La ventaja del dispositivo Kinect sobre las cámaras equivalentes separadas radica, además del coste, en un software interno en tubería que procesa los datos de la cámara de color, de la cámara infrarroja y los datos esqueletales. Esto lo hace más rápido que una implementación que integre dos cámaras separadas, además del extra que supone proveer la información de los esqueletos.

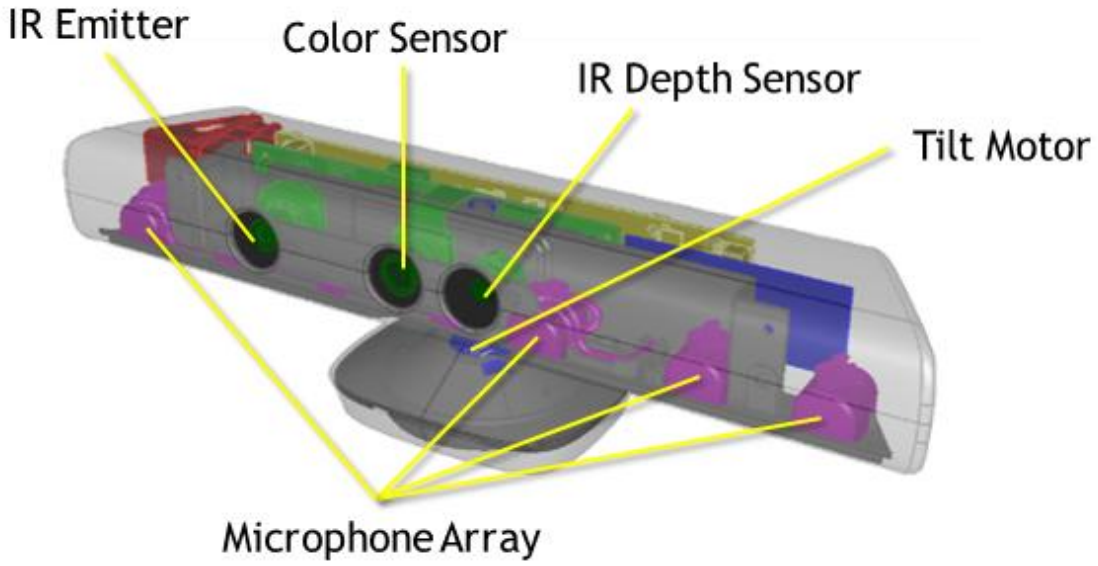


Figura 2: Disposición de los elementos internos del dispositivo Kinect.

Los componentes internos de Kinect son los siguientes:

- Una cámara RGB que almacena los datos en tres canales a una resolución de 1280x960 píxeles. Es la que hace posible capturar imágenes a color.
- Un emisor de infrarrojos (IR) y un sensor de profundidad por infrarrojos. El emisor lanza rayos de luz infrarroja y el sensor lee los rayos infrarrojos que se han reflejado de vuelta hacia el sensor. Estos rayos reflejados se convierten en información de profundidad, que indica la distancia entre un objeto y el sensor. Todo ello hace posible capturar imágenes de profundidad.
- Una serie de micrófonos para capturar sonido, compuesta por cuatro de ellos. Como hay cuatro micrófonos, es posible grabar audio además de localizar el origen del sonido y la dirección de la onda sonora.
- Un acelerómetro de tres ejes configurado para captar un rango de aceleración de 2G, donde G es la aceleración debida a la gravedad. Es posible usar el acelerómetro para determinar la orientación actual del dispositivo Kinect con respecto al suelo.

Kinect	Especificaciones
Ángulo de visión	Campo de visión de 43º en vertical y 57º en horizontal.
Rango de inclinación	±27º, en vertical.
Fotogramas/segundo	30 fotogramas por segundo para ambas cámaras.
Formato de audio	Modulación por impulsos codificados de 24 bits a 16 kHz.
Características de entrada de audio	Una serie de cuatro micrófonos con un convertidor analógico a digital de 24 bits y procesamiento de señal interno en Kinect, incluido cancelación de eco y eliminación de ruido.
Características del acelerómetro	Un acelerómetro de rangos 2G, 4G y 8G, configurado al rango de 2G y con un límite máximo de error de 1º.

Tabla 1: *Especificaciones de los componentes del dispositivo Kinect.*

A pesar de que el dispositivo Kinect cuenta con todas esas excelentes herramientas, cada una de ellas tiene sus propias limitaciones y especificaciones. Es recomendable conocerlas antes de intentar usar el dispositivo para analizar o realizar alguna acción que sus especificaciones impidan. Las limitaciones se encuentran en la Tabla 1, excepto las de la cámara de color y la cámara de infrarrojos. Esto se debe a que esas dos cámaras cambian su resolución automáticamente según la cantidad de

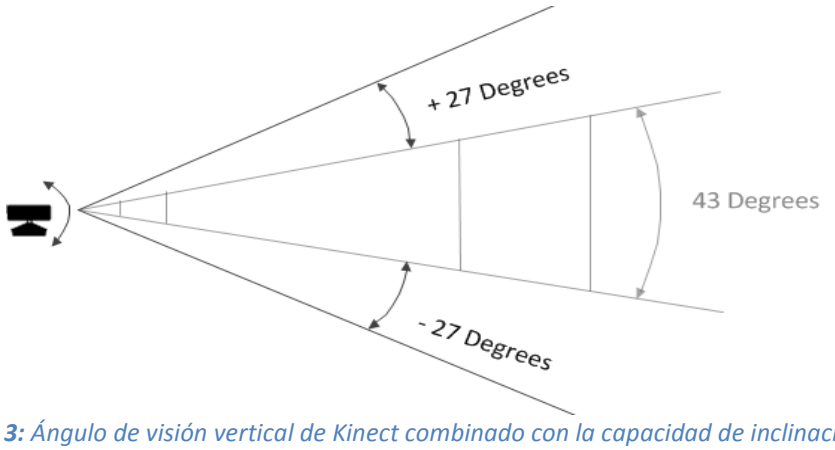


Figura 3: *Ángulo de visión vertical de Kinect combinado con la capacidad de inclinación.*

fotogramas por segundo que logren capturar. Las posibles resoluciones se especifican en las estructuras "ColorImageFormat" y "DepthImageFormat" respectivamente.

Conviene resaltar que, aunque el ángulo de visión vertical parezca muy limitado, Kinect posee una base mecanizada que puede inclinar el dispositivo hasta 27 grados hacia arriba o hacia abajo, como se puede ver en la Figura 3. Si se quiere, se puede comparar este ángulo de visión con el del ojo humano indicado en la Figura 4. Kinect posee un ángulo de visión vertical de 97 grados en total, mientras que el ojo posee una capacidad de visión cromática en 70 grados, aunque el límite de la visión total sea de 120 grados. Sin embargo, el dispositivo Kinect se queda atrás en lo referente al ángulo de visión horizontal, donde el ojo humano le aventaja enormemente.

Entorno de programación

Los entornos de programación que se pueden utilizar son limitados, ya que Kinect sólo puede funcionar en ciertos entornos específicos. Se pueden utilizar otros distintos si se pretende hacer uso de paquetes de controladores de código abierto, pero ese no es el caso. Por tanto, se utilizará el entorno de programación Microsoft Visual Studio 2010, compatible con todas las versiones del SDK de Kinect.

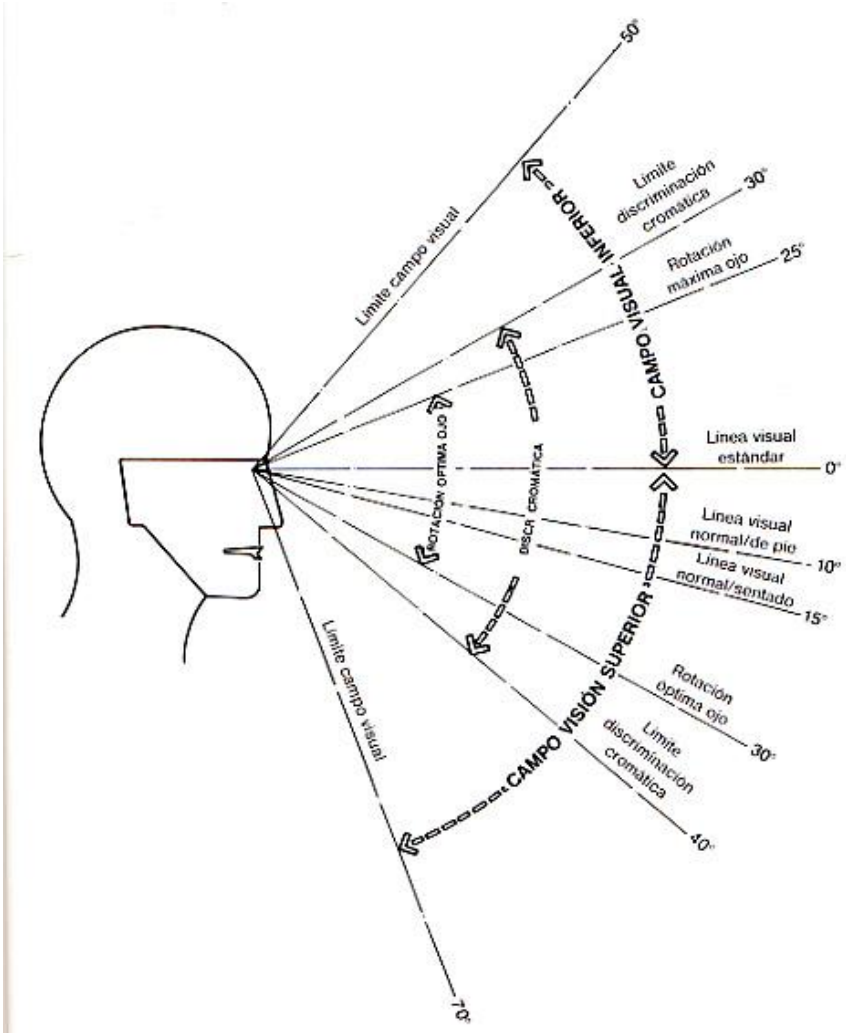


Figura 4: *Ángulo de visión vertical del ojo humano.*

SDK

El Kit de Desarrollo de Software para Kinect para Windows se encuentra actualmente disponible en la página web oficial de Microsoft[1] en su versión 1.7. Esta versión, actualizada en marzo de 2013, es la que se ha usado para la realización del proyecto en su totalidad, junto con el Kit de Herramientas para Desarrolladores de la misma versión.

En el apartado "SDK de Kinect - versión 1.7" de este mismo documento se realiza una amplia descripción de este Kit de Desarrollo de Software y de sus versiones anteriores.

Estructura de la memoria

El presente documento posee una estructura organizada en apartados. La distribución es la siguiente:

1. **Introducción:** Describe de forma preliminar el proyecto, así como las objetivos, la terminología y los medios que se van a usar a lo largo del mismo.
2. **SDK de Kinect - versión1.7:** Realiza una descripción del SDK de Kinect que se debe usar en toda aplicación que trabaje con el dispositivo, haciendo especial hincapié en algunas de sus funcionalidades.
3. **El seguimiento facial:** Describe con detalle la función de seguimiento facial presente como parte del Kit de Herramientas para Desarrolladores de Kinect, desde sus requisitos mínimos, pasando por las entradas necesarias y los resultados obtenidos.
4. **Análisis del código de ejemplo:** Realiza un estudio en profundidad y desglosa por clases el código de ejemplo utilizado como base para desarrollar la aplicación de la prueba de concepto. Se pretende que pueda ser usado como documentación de referencia por otros desarrolladores.
5. **Sistema de "escaparate caliente":** Determina el significado de "escaparate caliente" y expone la arquitectura de la prueba de concepto desarrollada, así como las pruebas a las que ha sido sometida, junto con sus resultados.
6. **Conclusiones y futuros trabajos:** Expone las conclusiones obtenidas a lo largo del trabajo y en vista de los resultados finales obtenidos. También trata de ponerlos en perspectiva y exponer qué otras líneas de trabajo se podrían seguir en un futuro.
7. **Bibliografía:** Contiene la compilación de todas las referencias usadas a lo largo de todo el trabajo.
8. **Anexos:** Otros aspectos del desarrollo que no se recogen en el cuerpo del documento, por ejemplo, la planificación y el presupuesto del mismo.

SDK de Kinect - versión 1.7

El SDK de Kinect para Windows es esencial para la planificación y programación de una aplicación, puesto que aporta los controladores necesarios para reconocer y manejar el dispositivo Kinect, los medios para usar dichos controladores de la forma correcta, y las funciones que procesarán los datos y los refinarán para que se puedan usar con mayor facilidad.

El SDK, además, da soporte a dos lenguajes distintos para poder realizar operaciones de más alto o más bajo nivel. Esto es, nos permite elegir si queremos manejar el programa con una configuración predeterminada o si preferimos indicarle cuándo y cómo realizar cada una de las acciones, como reservar memoria, por ejemplo. El lenguaje de alto nivel es C#, un lenguaje que se usa fundamentalmente en programas de Microsoft con base en Windows, mientras que el lenguaje de bajo nivel es C++, usado por multitud de compañías y sistemas operativos. Se pueden utilizar lenguajes adicionales para el diseño de la interfaz en el desarrollo de aplicaciones con interfaz gráfica, por ejemplo Visual Basic o XAML.

Para poder hacer uso del SDK al desarrollar aplicaciones, es necesario cumplir con una serie de requisitos tanto hardware como software. La mayoría de los requisitos hardware se pueden cumplir con un ordenador normal que tenga procesador de doble núcleo, pero es importante contar con un puerto USB 2.0 para conectar el dispositivo Kinect a él. Por supuesto, el propio dispositivo Kinect es otro de los requisitos hardware.

Como requisito software, se indica[2] que el SDK sólo puede ser utilizado en el entorno de programación Microsoft Visual Studio 2010 o 2012. Otros entornos no cuentan con soporte para el SDK. Si se van a utilizar funciones avanzadas, como reconocimiento de voz y análisis de profundidad en 3D, se requieren piezas de software adicionales que se encuentran listadas en la web de la Red de Desarrolladores de Microsoft [3].

La versión 1.7 del SDK de Kinect para Windows es una compilación de funciones resultado de sucesivas ampliaciones y optimizaciones por parte del desarrollador, Microsoft. El objetivo es proporcionar a los usuarios las herramientas que se necesitan para manejar el dispositivo Kinect de una forma accesible y segura. Para comprender mejor todas las funcionalidades que aporta el SDK, se van a describir las versiones que se han realizado del mismo.

SDK v. 1.0

Esta versión fue la primera que vio la luz tras un largo tiempo de desarrollo desde que se lanzó el dispositivo Kinect para Xbox 360 en noviembre de 2010. Si bien este dispositivo es compatible tanto con la Xbox 360 como con un PC, no existían controladores que permitieran obtener y manejar los datos del dispositivo.

Si se descuentan los paquetes de controladores de código abierto no oficiales y varias versiones beta del SDK, es con la versión 1.0 con la que finalmente se pueden desarrollar aplicaciones para PC que hagan uso completo del dispositivo Kinect. Esta versión, abierta para descarga en febrero de 2012, se acompañó del lanzamiento del dispositivo Kinect para Windows, creado específicamente para su uso con ordenadores, y con ciertas mejoras sobre el dispositivo para consola, como se ha descrito anteriormente.

Las principales funcionalidades que la versión 1.0 ofrece son:

- Control del flujo de datos de las dos cámaras; la cámara a color y la cámara infrarroja. De esta última se pueden obtener los datos de profundidad de los objetos en su rango de alcance.
- Reconocimiento y seguimiento del esqueleto de una persona, con mejoras significativas sobre las versiones beta.
- Capacidad de seguimiento activo del esqueleto de hasta dos personas. Se puede realizar un seguimiento posicional de hasta cuatro personas más.
- Reconocimiento de una cierta cantidad de gestos determinados con dedos o mano. Por ejemplo, el gesto de pasar una página.
- Control del conjunto de micrófonos de Kinect, así como reconocimiento de voz y de habla.
- Soporte para detección y manejo de desconexiones del dispositivo Kinect, así como de los casos de suspensión e hibernación del sistema.
- Soporte para conexión simultánea de hasta cuatro dispositivos Kinect, de los cuales sólo uno puede realizar un seguimiento de esqueletos.
- Soporte para el Modo Cercano, que permite enfocar y obtener los datos de profundidad de objetos que se encuentren a partir de 40 cm del dispositivo. No se puede realizar seguimiento de esqueleto en este modo.
- Capacidad de construir aplicaciones tanto en 32 bits como en 64 bits.

Además de las funcionalidades anteriores, esta versión del SDK cuenta con una amplia librería de programas de ejemplo, tanto ejecutables como códigos fuente para su consulta.

SDK v. 1.5

El lanzamiento de esta versión del SDK vino acompañado del lanzamiento del Kit de Herramientas para Desarrolladores de la misma versión. Este paquete cuenta con las clases y funciones avanzadas, así como con los programas de ejemplo que antes se encontraban incluidos dentro del paquete del SDK.

El SDK, por otro lado, sigue manteniendo todas las clases y funciones necesarias para el manejo del dispositivo Kinect, sólo se han separado aquellas construidas en base a las anteriores, no se ha perdido ninguna funcionalidad. Esta organización permite actualizar por separado el Kit de Herramientas para Desarrolladores cuando no se ha producido ningún cambio en lo que es el núcleo del controlador.

Aparte de la novedad de la separación en dos paquetes, esta versión del SDK trajo también considerables mejoras. Una de ellas fue la capacidad de seguimiento de esqueletos en Modo Cercano, una mejora lógica si se tiene en cuenta una de las nuevas funcionalidades de esta versión, el seguimiento de esqueletos en Modo Sentado, que permite reconocer la mitad superior del esqueleto de una persona que está sentada. En realidad, este modo también se puede aplicar a personas que estén de pie, pero tiene mayor coste de procesamiento y sólo obtiene la mitad superior del cuerpo.

La funcionalidad de reconocimiento del habla se ha mejorado para ampliar el soporte de varios lenguajes y dialectos más; el inglés no estadounidense, el español de España y de México, el francés de Francia y de Canadá, el italiano y el japonés.

Por último, se ha añadido una nueva funcionalidad, la detección y el seguimiento facial. Esta funcionalidad permite obtener la posición de la cara a partir de un esqueleto, independientemente de si es en Modo Normal o Modo Sentado. Con esa posición, el dispositivo Kinect es capaz de obtener una serie de puntos faciales que determinan el contorno facial, la posición de los ojos y la boca, y la forma de la nariz.

Mientras el SDK de Kinect se mantuvo en esta versión, se dieron dos actualizaciones más del Kit de Herramientas para desarrolladores, con la inclusión de nuevo contenido en el mismo. Se puede destacar el gran número de ejemplos que se añadieron, así como un programa para grabar y reproducir vídeo de Kinect y mejoras en la clase avanzada para el seguimiento facial.

SDK v. 1.6

Esta versión del SDK incluye soporte para el entorno de programación Microsoft Visual Studio 2012, pudiendo así aprovechar las mejoras que esto implica, no sólo para la aplicación resultante, sino para el proceso de programación en sí.

En lo referente a contenido, esta nueva versión del SDK introduce APIs para acceder a los datos del acelerómetro, pudiendo así saber la orientación del sensor sobre el suelo, APIs para convertir el espacio de coordenadas 3D a píxeles de una imagen, APIs para el control de la configuración de la cámara de color, APIs para obtener el flujo de la cámara de infrarrojos, útil para obtener imágenes en escala de grises si no hay suficiente luz para la cámara de color, y un API para controlar el emisor de infrarrojos, ya que antes siempre estaba activo mientras el dispositivo estuviera encendido, mientras que ahora se puede apagar y encender a voluntad.

Aparte de las APIs, se ha introducido una nueva función que permite obtener datos de profundidad más allá del límite normal de cuatro metros, aunque se advierte de que la información se degrada rápidamente con la lejanía más allá de este límite, de modo que los datos no son tan fiables. También se ha añadido un paquete de lengua alemana para la funcionalidad de reconocimiento del habla.

Por último, se ha añadido soporte para máquinas virtuales. Anteriormente, no se podía utilizar el dispositivo Kinect si Windows se estaba ejecutando en una máquina virtual. Ahora, el SDK soporta las siguientes máquinas virtuales: Microsoft Hyper-V, VMWare y Parallels. Esto hace que se amplíe enormemente la utilidad del dispositivo Kinect, ya que se puede utilizar en cualquier sistema que pueda ejecutar Windows en una de esas máquinas virtuales. Aunque debe tenerse en cuenta que el rendimiento puede disminuir debido al consumo de recursos por la propia máquina virtual. Sólo se puede utilizar un dispositivo Kinect por ordenador si se está utilizando una máquina virtual.

SDK v. 1.7

Esta versión del SDK sólo contiene algunas mejoras funcionales, no añade ninguna funcionalidad básica. Así, se ha mejorado el soporte de detección de desconexiones de dispositivos Kinect y se ha simplificado el manejo de dichos eventos. También se ha añadido un paquete nuevo de gestos reconocibles por la funcionalidad de reconocimiento de gestos, entre los que destacan el de Pulsar para Seleccionar y el de Agarrar para Desplazarse.

En el Kit de Herramientas para Desarrolladores, se ha añadido una gran cantidad de ejemplos de código. Algunos de ellos pueden resultar muy interesantes en ciertos círculos, ya que demuestran cómo procesar imágenes del dispositivo Kinect con MatLab o con OpenCV.

Sin embargo, la gran joya de esta versión del Kit de Herramientas para Desarrolladores es una funcionalidad nueva, Kinect Fusion, que permite hacer un escaneado tridimensional de un objeto o una habitación. Este escaneado genera un detallado modelo 3D de la escena y Kinect Fusion permite interactuar con éste por medio de Kinect de la forma habitual. Las posibilidades de esta funcionalidad nueva son enormes, ya que permite crear con facilidad una aplicación de realidad aumentada.

El seguimiento facial

El seguimiento facial es una funcionalidad que se introdujo en el Kit de Herramientas para Desarrolladores de Kinect en la versión 1.5[2]. Como se ha mencionado anteriormente, esta funcionalidad es capaz de identificar la cara de una persona y obtener una malla de puntos en tres dimensiones que perfilan su forma.

Sin embargo, ésta no es su única capacidad. Para obtener la malla de puntos, Kinect debe obtener primero una serie de parámetros adicionales. Todos ellos están disponibles para ser usados por un programador habilidoso, desde lo inclinada que está la cabeza hasta lo amplia que es la sonrisa o lo alzadas que están las cejas. La funcionalidad de seguimiento facial tiene un potencial enorme.

Requisitos mínimos

- Ordenador con un procesador de doble núcleo a 2.66-GHz o más.
- Tarjeta gráfica compatible con Windows 7 o Windows 8, y que soporte Microsoft® DirectX® 9.0c.
- 2 GiB de memoria RAM
- Dispositivo Kinect—se indica que el que se compra en tiendas, ya que viene con un USB especial y cable de corriente.

Sistema de coordenadas

El sistema de coordenadas que se usa para representar la malla de puntos faciales tiene el origen en el centro de la lente externa de la cámara de color del dispositivo Kinect. El eje Z apunta hacia fuera, en el mismo sentido que está apuntando la cámara, mientras que el eje Y apunta hacia arriba[4]. La Figura 5 muestra claramente el espacio de coordenadas de la cámara y cómo se representaría la malla de puntos.

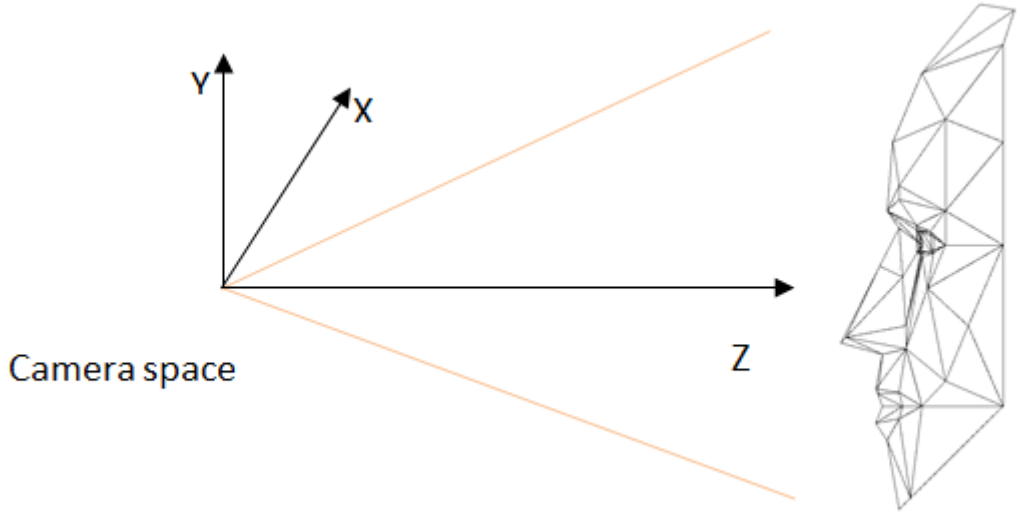


Figura 5: *Espacio de coordenadas y malla facial.*

Imágenes de entrada

El seguimiento facial necesita como entrada la imagen de la cámara a color y los datos de profundidad de la cámara infrarroja. Sólo a partir de estos dos, se puede obtener la malla de puntos de una cara. Sin embargo, también se puede ayudar al proceso limitando la zona a buscar de dos formas, utilizando también el esqueleto de la persona como entrada o bien proporcionando un rectángulo dentro del cual está la cara que nos interesa. Pueden usarse incluso ambos a la vez para una detección aún más rápida.

Salida del seguimiento facial

El seguimiento facial produce una serie de datos de salida sobre la cara que haya encontrado. Si es que ha encontrado una, claro está. Todos estos datos se recogen en una instancia, y son los siguientes:

- Estado del seguimiento
- Puntos de la cara en 2D
- Posición de la cabeza en 3D
- Unidades de Forma (Shape Units)
- Unidades de Animación (Animation Units)

El estado del seguimiento indica si el seguimiento ha tenido éxito o no. Es decir, si se ha logrado encontrar la cara para analizarla. Si no se ha podido encontrar, el resto de datos serán nulos.

Por otro lado, los puntos de la cara en 3D no se obtienen directamente, pero se pueden calcular a partir del resto de los datos. La propia API realiza este proceso, no es necesario realizarlo manualmente[4].

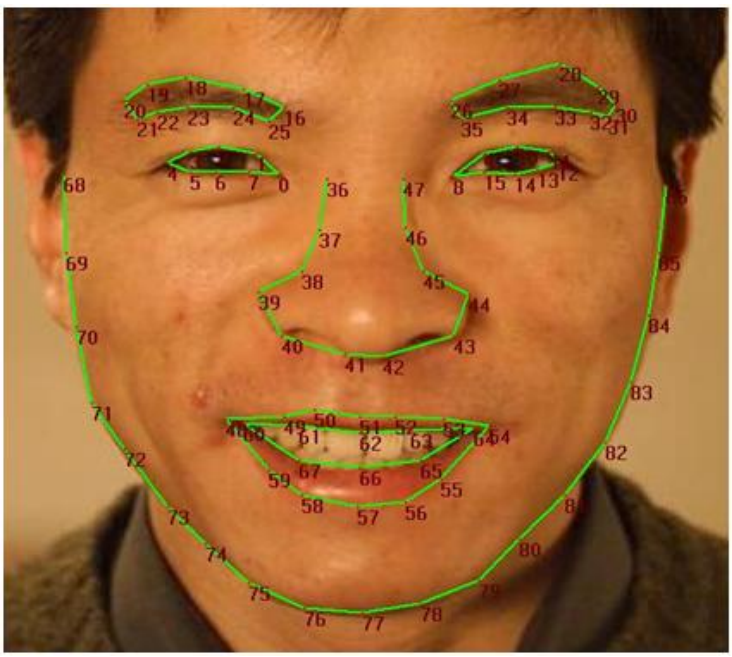


Figura 6: *Supuesta distribución de los primeros 87 puntos faciales.*

Puntos de la cara en 2D

Como resultado del seguimiento facial se obtienen una serie de puntos sobre la cara detectada. Según la documentación oficial de Kinect[4], los puntos detectados son 100 en total, 87 de los cuales se pueden ver señalados en la Figura 6. Sin embargo, en la práctica, al obtener los puntos de la cara de una persona se reciben 121 puntos. En la Figura 7 se puede apreciar la gran diferencia entre los puntos supuestos y los obtenidos.

Figura 7: *Malla de puntos resultante del seguimiento facial.*

Posición de la cabeza en 3D

Para definir la posición de la cabeza, se usan las coordenadas de un punto central y el valor de tres ángulos denominados "roll", "pitch" y "yaw" por sus términos en inglés[4]. Traducidos al español sería algo parecido a "inclinación", "cabeceo" y "viraje", pero se usarán los términos en inglés por sencillez. Una explicación gráfica de estos ángulos se puede encontrar en la Figura 8.

Las coordenadas del punto central de la cabeza se toman con respecto al mismo sistema de coordenadas que se usan para los puntos faciales. Es probable que este punto central coincida con la articulación "Head" (Cabeza) del esqueleto que obtiene el dispositivo Kinect de la persona, pero no se especifica.

Volviendo a los ángulos que determinan la posición de la cabeza, los valores que pueden tener estos ángulos se concretan en la Tabla 2. También se incluye el rango de valores en los que funciona el seguimiento facial.

Ángulo	Valor
Roll *0 = neutral*	-90 = Cabeza horizontalmente en el lado derecho del sujeto. +90 = Cabeza horizontalmente en el lado izquierdo del sujeto. El seguimiento facial funciona cuando este ángulo tiene una diferencia inferior a 90 sobre el neutral, pero funciona mejor con menos de 45.
Pitch *0 = neutral*	-90 = Mirando verticalmente hacia el suelo. +90 = Mirando verticalmente hacia el techo. El seguimiento facial funciona cuando este ángulo tiene una diferencia inferior a 20 sobre el neutral, pero funciona mejor con menos de 10.
Yaw *0 = neutral*	-90 = Cabeza girada hacia el hombro derecho del sujeto. +90 = Cabeza girada hacia el hombro izquierdo del sujeto. El seguimiento facial funciona cuando este ángulo tiene una diferencia inferior a 45 sobre el neutral, pero funciona mejor con menos de 30.

Tabla 2: *Rango de valores de los ángulos roll, pitch y yaw.*

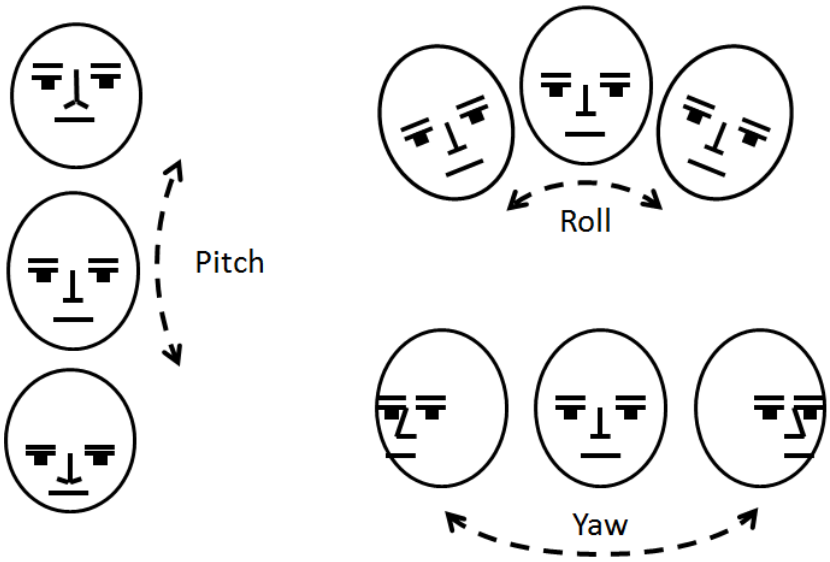


Figura 8: *Ángulos roll, pitch y yaw.*

Unidades de Forma

Las Unidades de Forma (UF) sirven para estimar la forma específica de la cara del individuo; la posición neutral de la boca, los ojos, las cejas, etc. Cada UF afecta a una serie de vértices determinados de la malla facial, que se desplazarán según el valor que tenga la UF. La lista de las UFs existentes es la siguiente, con sus nombres en inglés:

- Head height
- Eyebrows vertical position
- Eyes vertical position
- Eyes, width
- Eyes, height
- Eye separation distance

- Eyes vertical difference
- Nose vertical position
- Mouth vertical position
- Mouth width
- Chin width

Con estas medidas, se puede obtener la forma neutral de la cara, sin expresiones. Por esta razón, se podrían usar de varias maneras, como aplicar esos rasgos faciales propios a un avatar que represente a la persona, por ejemplo.

Unidades de Animación

Las Unidades de Animación (UA) se utilizan como diferencias que se aplican sobre la cara en estado neutral para obtener una expresión facial. Los valores pueden estar entre 1 y -1, donde 0 indica neutralidad. Usadas en conjunto con la malla facial y las UFs, se puede obtener un modelo tridimensional de cualquier cara posible, independientemente de la expresión[4]. La Tabla 3 muestra información más detallada sobre las UAs.

Nombre y número de UA	Ilustración en avatar	Interpretación de valores
Cara neutral (Todas las UAs a 0)		No aplicable
UA0 - Upper Lip Raiser		0 = Neutral, cubriendo los dientes. 1 = Enseñando los dientes del todo. -1 = Labio empujado hacia abajo todo lo posible.
UA1 - Jaw Lowerer		0 = Boca cerrada. 1 = Boca completamente abierta. -1 = Boca cerrada, igual que en 0.

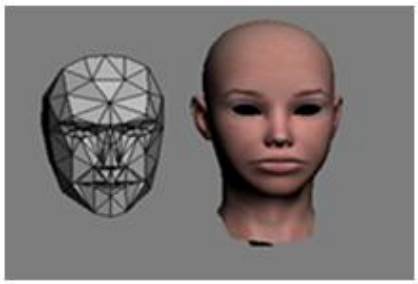
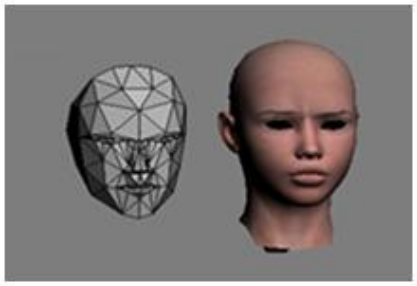
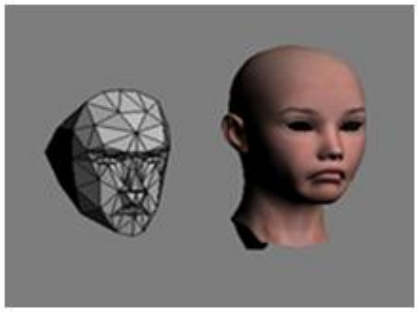
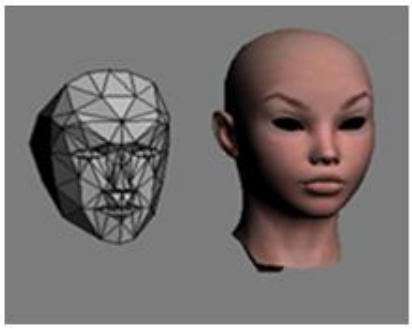
UA2 - Lip Stretcher		0 = Neutral. 1 = Labios completamente estirados (sonrisa del Joker). -0,5 = Labios redondeados -1 = Labios totalmente redondeados (gesto de besar).
UA3 - Brow Lowerer		0 = Neutral. 1 = Ceño totalmente bajado, hasta el límite de los ojos. -1 = Ceño alzado al máximo.
AU4 - Lip Corner Depressor		0 = Neutral. 1 = Gesto muy triste (labios hacia abajo). -1 = Sonrisa muy feliz.
AU5 - Outer Brow Raiser		0 = Neutral. 1 = Cejas alzadas como en un gesto de mucha sorpresa. -1 = Cejas bajas como en una cara de mucha tristeza.

Tabla 3: *Nombre, ilustración y valores posibles de las UAs.*

Influencia de CANDIDE versión 3

En el seguimiento facial, el modelo de la cara que se genera, así como el sistema de Unidades de Forma y Unidades de Animación, se realizó basándose en el modelo facial CANDIDE-3[4]. Este modelo facial, originalmente descrito por M. Rydfalk[5] y posteriormente mejorado en tres ocasiones[6], posee una malla facial similar a la facilitada por el dispositivo Kinect y una serie de UFs y UAs que no se diferencian mucho de las usadas por el seguimiento facial.

CANDIDE es una malla facial parametrizada que se diseñó específicamente para la codificación de caras humanas en forma de modelos. Su relativamente bajo número de polígonos (sobre los cien) permite una reconstrucción rápida de la cara con una capacidad de computación moderada.

Unas *Unidades de Acción* (*UA*) globales y locales permiten controlar a CANDIDE. Las *UA*s globales corresponden a su rotación con respecto a tres ejes, mientras que las *UA*s locales controlan las imitaciones de la cara de forma que se puedan obtener distintas expresiones[6].

Como se puede imaginar, estas características de bajo coste computacional y facilidad del manejo del modelo hacen que CANDIDE sea una opción ideal para su manejo con Kinect. De ahí que se usara como base para crear el modelo usado en seguimiento facial, que se adapta a las necesidades del dispositivo.

Análisis del código de ejemplo

Los programas de ejemplo que se recogen en el Kit de Herramientas para Desarrolladores son muy útiles tanto para la iniciación al uso del dispositivo Kinect como para el aprendizaje de nuevas técnicas y funcionalidades una vez se domina lo básico del manejo y sus peculiaridades.

Sin embargo, esta base puede ser difícil de adquirir, puesto que la documentación es algo oscura con respecto a cómo empezar, obviando algunos pasos necesarios o no indicando muy bien qué se debe hacer o por qué. En esos casos, el código de ejemplo también puede resultar muy opaco, dependiendo del programa cuyo código fuente se esté usando.

En el caso que nos ocupa, el código de ejemplo utilizado fue el de una programa llamado **FaceTrackingBasics-WPF**, que demuestra y aplica los métodos básicos de seguimiento facial en lenguaje de alto nivel. El comienzo fue desde cero, sólo conociendo los detalles generales del lenguaje de programación C#, de modo que se tuvo que realizar una tarea de análisis del código de ejemplo para discernir las tareas de cada clase, las interacciones entre ellas, y el flujo de la ejecución. Con ello se aprendieron las funciones básicas necesarias para el manejo del dispositivo Kinect y se localizaron las funciones avanzadas que podrían utilizarse de cara al desarrollo de la futura aplicación.

Por tanto, se pretende documentar los resultados del proceso de análisis para su uso como referencia por futuros desarrolladores que tengan que comenzar el proceso de aprendizaje en el uso del dispositivo, así como su uso en forma de listado de recursos que se pueden utilizar para resolver otros problemas más avanzados.

El código de ejemplo se divide en tres paquetes, que se analizarán de forma separada para mayor sencillez.

Paquete "Microsoft.Kinect.Toolkit"

Este paquete, como se podría deducir a partir del nombre, contiene clases que se encuentran en el Kit de Herramientas para Desarrolladores. Este paquete en concreto cuenta con todo el código necesario para realizar una gestión de las conexiones y desconexiones de dispositivos Kinect, además de la asignación de un sensor no ocupado a un proceso que lo requiera.

Esto puede ser muy útil en caso de usar varios sensores a la vez o si se quiere diseñar una aplicación que soporte hibernaciones o suspensiones del sistema. También puede usarse en aplicaciones que vayan destinadas a terceros, para así evitar problemas imprevistos en caso de que esas terceras personas ejecuten la aplicación sin el dispositivo Kinect conectado u otros similares.

Si se pretende hacer uso de esta funcionalidad, se puede usar este paquete al completo, ya que posee todo lo necesario para realizar la gestión y además contiene una interfaz de usuario que se puede utilizar para comprobar los estados de las conexiones con los dispositivos. Esta interfaz es opcional y no afecta al rendimiento de la funcionalidad principal. De hecho, las tres clases que lo componen pueden eliminarse sin afectar al resto del paquete. Por esta razón, no se han incluido en el análisis que se realiza a continuación a las clases KinectSensorChooserUI.xaml, KinectSensorChooserUiViewModel.cs y RelayCommand.cs.

Como advertencia, se ha observado que este paquete puede provocar un error en ciertas ocasiones. Si se está ejecutando una aplicación en modo Debug con Microsoft Visual Studio 2010 y se presiona el botón de "Detener ejecución" durante la misma, podría ocurrir que el dispositivo Kinect quedase bloqueado y no respondiese ante una nueva ejecución de la clase KinectSensorChooser. En tal caso, es suficiente con desenchufar el conector USB de Kinect del PC y volverlo a conectar. Ni siquiera es necesario parar la ejecución del nuevo programa.

CallbackLock.cs

Esta clase contiene la funcionalidad de un cerrojo (lock) estándar, útil para programas multihilo. En este paquete, se utiliza de forma extendida para permitir la gestión de varios dispositivos a la vez.

ContextEventWrapper.cs

Esta clase contiene en realidad la declaración de tres clases distintas. La clase ContextEventWrapper es un envoltorio para poder manejar dos tipos de eventos concretos de forma sencilla. Se usa en la clase KinectSensorChooser para envolver al evento KinectChangedEvent y el evento PropertyChangedEvent, ambos obtenidos del dispositivo Kinect cuando se producen ciertos cambios.

Las clases ContextEventHandlerWrapper y ContextHandlerPair se utilizan como objetos instanciables por la clase anterior para almacenar datos.

KinectChangedEventArgs.cs

Esta clase es una clase instanciable que contiene los datos que acompañan a un evento del tipo KinectChangedEvent.

KinectSensorChooser.cs

La clase principal del paquete. Obtiene un dispositivo Kinect libre y notifica de ello al proceso que le haya ordenado buscar dispositivos. También monitoriza los cambios relacionados con el dispositivo obtenido, para poder buscar un dispositivo nuevo en caso de que el actual se haya desconectado, o volver a obtener el anterior una vez se reconecte. No tiene autoridad para quitarle a otra aplicación un sensor que esté controlando.

Esta clase se comunica con los procesos interesados a base de eventos de dos tipos, KinectChangedEvent y PropertyChangedEvent; de modo que un proceso que necesite hacer uso de un dispositivo Kinect tendrá que poseer los manejadores adecuados para procesar estos eventos.

ThreadSafeCollection.cs

Esta clase es una implementación de la clase IList con cerrojos en todas las operaciones. IList es una clase base para crear listas genéricas de datos u objetos, por lo que añadirle cerrojos a las operaciones la hace adecuada para ser utilizada en procesos multihilo. Esto se debe a que garantiza que no puede accederse al contenido simultáneamente por parte de dos hilos distintos, evitando así los múltiples problemas que se derivan de ello.

Paquete "Microsoft.Kinect.Toolkit.FaceTracking"

Este paquete contiene unas clases más específicas del Kit de Herramientas para Desarrolladores que el paquete anterior. Se encuentran en un paquete distinto debido a que no son clases de funcionalidad general que pueden servir para cualquier aplicación; si se usan, será en aplicaciones más especializadas.

Como el nombre en inglés indica, las funcionalidades recogidas en este paquete tienen relación con la detección y el seguimiento facial de personas. Las clases presentes en el paquete proporcionan estructuras de datos y un API para obtener la localización de la cara y ciertos puntos localizados a lo largo de su superficie, pero no proporciona prácticamente ningún método para filtrar o manejar los datos. La mayor parte del manejo debe programarlo uno mismo.

Si se pretende hacer uso de la funcionalidad de seguimiento facial, y sobre todo si se piensa obtener los distintos puntos que el dispositivo Kinect puede detectar a lo largo de la cara, se debe utilizar este paquete al completo. Ninguna de las clases es opcional.

EnumIndexableCollection.cs

Una clase que permite almacenar una colección de datos y acceder a los elementos simplemente por el valor de un índice. Internamente, la colección es de sólo lectura, de modo que la clase actúa más como un índice que como una colección en sí. Al obtener los puntos faciales, se envían contenidos en una instancia de esta clase.

FaceModel.cs

Es la clase que se encarga de almacenar el modelo tridimensional de los puntos de la cara en forma de una malla de triángulos. En realidad se trata de un API que accede al modelo almacenado internamente en el dispositivo Kinect. Con esta clase se pueden obtener los puntos de la cara en tres dimensiones, esos mismos puntos pero proyectados en dos dimensiones de forma que coincidan con la imagen de la cámara RGB, y el orden de los triángulos que conforman la malla facial.

Esta clase probablemente nunca se llegue a usar en una programación propia a no ser que se quieran obtener datos de los puntos faciales con una configuración distinta a la de la cámara del dispositivo Kinect. En cualquier otro caso, se usará de forma implícita a través de los métodos disponibles en la clase FaceTrackFrame.

FaceTracker.cs

La clase central de este paquete. Contiene los métodos que permiten identificar la cara de una persona a partir de su esqueleto y realizar un seguimiento de la misma. Al igual que el método anterior, se trata de un API que accede a los datos contenidos en una clase nativa del dispositivo Kinect. El acceso a estos datos se realiza a través de la clase FtInterop.

El uso de esta clase radica en la asignación de una instancia de la misma a cada esqueleto sobre el que se quiera realizar el seguimiento. En teoría, podría usarse sólo una instancia por cada dispositivo Kinect del que se reciben datos, pero se pierde mucho rendimiento si cada instancia realiza el seguimiento a más de un esqueleto, ya que tendría que cambiar los datos internos cada vez que cambie de uno a otro. A la instancia se le pasan los datos necesarios por medio del método Track(), del que se obtiene una instancia de la clase FaceTrackFrame con todos los datos faciales del individuo.

Sólo se pueden obtener datos faciales de personas que tengan un seguimiento activo de esqueleto; aquellas con seguimiento pasivo no serán reconocidas por el seguimiento facial. También se advierte que la malla facial puede tener notables errores si las condiciones no son ideales, ya sea poca luz, abundante presencia de vello facial u objetos oclusivos, o simplemente que la cara se encuentre muy inclinada o ladeada. En muchas ocasiones se pierde la malla facial si se deja de ver uno de los ojos.

FaceTrackFrame.cs

Esta clase contiene los datos faciales que se hayan obtenido después del seguimiento facial exitoso en un fotograma. Es la clase de la que se obtendrán la mayoría de los datos, aunque, como las dos clases anteriores, se trata de un API que accede a datos internos.

La mayor utilidad de esta clase radica en la obtención de los puntos que forman la malla facial (aunque en realidad se obtienen de la clase FaceModel), tanto en 2D como en 3D, pero también puede tener otras utilidades. Por ejemplo, el método GetAnimationUnitCoefficients() permite obtener los coeficientes de las unidades de animación. Estas unidades de animación indican cuánto está abierta la boca, cuán amplia es la sonrisa, cuán alzadas están las cejas, etc. Las unidades de animación se explican con detalle en el apartado anterior: "Seguimiento facial".

FtInterop.cs

Una clase que aglomera varias clases API. Se compone prácticamente de declaraciones de métodos que no tienen contenido, por lo que probablemente se traten de metadatos asociados a las clases nativas del dispositivo Kinect.

Image.cs

Esta clase sirve para almacenar y manejar imágenes de forma genérica. Se podría pensar que es un envoltorio para otras clases de imagen cuyos datos se almacenan en otro sitio, pero también tiene la capacidad de reservar memoria y almacenar datos ella misma. Por tanto, puede cumplir la función de actuar como referencia a una imagen externa o actuar como una instancia de una imagen. Las clases FaceTracker y FtInterop la usan de ambas formas.

Utils.cs

Esta clase contiene las declaraciones de dos enumerados y cinco estructuras que se usan con asiduidad por clases de todo el paquete. Uno de los enumerados contiene los nombres de las unidades de animación, mientras que el otro asigna un nombre a ciertos puntos de la malla facial.

Entre las estructuras hay dos que definen un punto en dos dimensiones, una que define un rectángulo en dos dimensiones a partir de cuatro puntos pasados por índice en lugar de por coordenadas, otra que define un vector en tres dimensiones (puede usarse para definir un punto tridimensional también) y una última estructura que define un triángulo de la misma forma que el rectángulo anterior.

Entre ellas, las más útiles son PointF (punto en dos dimensiones con parámetros de tipo float) y Vector3DF (vector/punto en tres dimensiones con parámetros de tipo float), ya que son las que se usan para obtener los puntos faciales en dos y tres dimensiones respectivamente.

Paquete "FaceTrackingBasics-WPF"

Este paquete es el que contiene la funcionalidad del programa. Al tratarse de un WPF, el paquete además contiene el código necesario para mostrar la interfaz de usuario y realizar todas las operaciones relacionadas. Para ello, se basa en pares de archivos de extensión .xaml y .cs donde el primero define todas las características de los elementos estáticos de la interfaz y contiene el nombre y la referencia de los elementos dinámicos, mientras que el segundo se encarga del procesamiento interno y la actualización de los elementos dinámicos.

La funcionalidad completa de este paquete es sencilla. Se genera una ventana donde se muestra el vídeo capturado con la cámara de color del dispositivo Kinect. Sobre esta imagen, fotograma a fotograma, se dibuja la malla de puntos faciales de todas las personas presentes a las que pueda realizarles el seguimiento. Debido a las limitaciones actuales, el máximo es dos con un solo ordenador.

Este paquete contiene tres pares xaml-cs, de los cuales sólo dos son funcionales. Si se comprende su funcionamiento, el paquete puede resultar una base excelente para expandir su funcionalidad, ya que la forma en que está estructurado lo dota de una buena estabilidad.

App.xaml y App.xaml.cs

Este par xaml-cs está vacío. Ni la parte de interfaz de usuario ni la de control interno contienen métodos o datos.

MainWindow.xaml

Este archivo contiene los detalles de la interfaz de usuario que viene a ser la ventana que se muestra al ejecutar el programa. La estructura del archivo es muy parecida a la de un .xml estándar y se puede descifrar su estructura con relativa facilidad.

```
<Window
    x:Class="FaceTrackingBasics.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/
                                    2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:FaceTrackingBasics"
    Title="Face Tracking Basics"
    Closed="WindowClosed"
    Height="735" Width="770"
    >

<Window.Resources>
    <SolidColorBrush x:Key="MediumGreyBrush"
                                    Color="#ff6e6e6e"/>
    <SolidColorBrush x:Key="KinectPurpleBrush"
                                    Color="#ff52318f"/>
    <SolidColorBrush x:Key="KinectBlueBrush"
                                    Color="#ff00BCF2"/>
    <Style TargetType="{x:Type Image}">
        <Setter Property="SnapsToDevicePixels"
                                    Value="True"/>
    </Style>
</Window.Resources>
```

En conjunto, la interfaz que se declara es una ventana de 770x735 píxeles con el título "Face Tracking Basics" y que al presionar el botón de cerrar llama al método interno (en el archivo .cs) WindowClosed(). Después de esto, se declaran unos colores que serán usados más adelante, y se llega a la sección de código que declara la distribución de los elementos.

```
<Grid Name="layoutGrid" Margin="10 0 10 10">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <DockPanel Grid.Row="0" Margin="0 0 0 20">
        [...]
    </DockPanel>
    <Viewbox Grid.Row="1" Stretch="Uniform"
                            HorizontalAlignment="Center">
        <Grid Name="MainGrid" Width="640" Height="480">
            <Image Name="ColorImage"/>
            <local:FaceTrackingViewer
                            x:Name="faceTrackingViewer" />
        </Grid>
    </Viewbox>
</Grid>
```

Se declaran dos filas. La superior (número 0) contiene un par de imágenes y un texto, mientras que la inferior (número 1) contiene un objeto de imagen con el nombre de "ColorImage". Este es un elemento dinámico cuyos datos pueden ser alterados por el procesamiento interno, y que mostrará los fotogramas capturados por la cámara de color del dispositivo Kinect. Además de esta imagen, en la misma fila, se incluye una referencia a la clase FaceTrackingViewer, de modo que los elementos de su interfaz de usuario se superpondrán sobre la imagen.

MainWindow.cs

La clase que controla el funcionamiento interno de su interfaz de usuario asociada, MainWindow.xaml. Este funcionamiento es sencillo, puesto que tiene unas funciones limitadas y diferenciadas. Los métodos que contiene comprenden la inicialización de la ventana de la interfaz junto con el inicio del dispositivo Kinect, el cambio de sensor en caso de que KinectSensorChooser le asigne uno nuevo, y la redirección del flujo de la cámara de color para que se muestre en la imagen de la interfaz de usuario. Un último método se encarga de eliminar la interfaz de usuario al pulsar el botón de cerrar.

La mejor forma de comprender el funcionamiento de una clase es observar el flujo de la ejecución, es decir, la cadena de llamadas que los métodos se hacen los unos a los otros hasta el final de la ejecución. En programas normales, este flujo es claramente lineal, aunque los más complejos incluirán ejecuciones paralelas en hilos, lo cual es esencialmente añadir un flujo de ejecución extra al programa. Sin embargo, las aplicaciones que deben trabajar con sensores tienen un funcionamiento distinto. Los sensores utilizan unos avisos denominados "Eventos" que permiten que la aplicación se mantenga a la espera hasta que le lleguen dichos avisos, momento en que ha ocurrido algo en el sensor y el programa debe procesarlo de forma acorde.

El dispositivo Kinect produce eventos cada vez que cada uno de sus flujos de datos tiene un fotograma listo, aunque estos flujos hay que activarlos explícitamente antes de que empiecen a producir fotogramas. También produce eventos cada vez que una de sus propiedades cambia. KinectSensorChooser se encarga de monitorizar estos eventos y, en caso de que un dispositivo Kinect se pierda, asignar otro a ese proceso. El proceso es notificado con un evento nuevo, que le indica que su dispositivo ha cambiado. Esta clase, MainWindow.cs tiene métodos para procesar algunos de estos eventos, como se verá más adelante.

Una vez se conoce cómo funcionan los eventos, discernir el flujo de ejecución es sencillo. Nada más iniciarse la aplicación, el método que se ejecuta es el constructor de la clase; MainWindow().

```
public MainWindow()
{
        InitializeComponent();

        var faceTrackingViewerBinding = new Binding("Kinect")
                                        { Source = sensorChooser };
        faceTrackingViewer.SetBinding(
                            FaceTrackingViewer.KinectProperty,
                            faceTrackingViewerBinding);

        sensorChooser.KinectChanged += SensorChooserOnKinectChanged;

        sensorChooser.Start();
}
```

Lo primero que se realiza en el método es inicializar la interfaz de usuario con todo lo que eso conlleva; se crea un objeto de tipo imagen "ColorImage" y se crea una instancia de FaceTrackingViewer con el nombre "faceTrackingViewer". Después, el método enlaza una propiedad de esta clase con una instancia de KinectSensorChooser, de modo que a FaceTrackingViewer también le notifiquen los eventos producidos por KinectSensorChooser, pero de forma especial. El siguiente paso es indicarle a KinectSensorChooser que esta clase procesará los eventos de tipo "KinectChanged" con el método SensorChooserOnKinectChanged(). Por último, KinectSensorChooser empieza a controlar los dispositivos Kinect y a asignarlos según convenga.

En resumen, con el constructor se consiguen tres cosas:

1. Iniciar la interfaz de usuario.
2. Que tanto esta clase como FaceTrackingViewer obtengan notificaciones cuando se cambie el dispositivo Kinect que está en uso.
3. Iniciar la búsqueda y obtención de dispositivos Kinect.

El flujo de ejecución normal acaba aquí, puesto que la llamada a KinectSensorChooser no devuelve nada. El programa entra en espera hasta que se obtenga un evento de tipo "KinectChanged" que le indica que ha obtenido acceso a un dispositivo Kinect. Cuando se produzca, el evento se procesará con el método SensorChooserOnKinectChanged().

```
private void SensorChooserOnKinectChanged(object sender,
KinectChangedEventArgs kinectChangedEventArgs)
{
       KinectSensor oldSensor = kinectChangedEventArgs.OldSensor;
       KinectSensor newSensor = kinectChangedEventArgs.NewSensor;

       if (oldSensor != null)
       {
           oldSensor.AllFramesReady -=
                           KinectSensorOnAllFramesReady;
           [...]
       }

       if (newSensor != null)
       {
           try
           {
               newSensor.ColorStream.Enable(
                     ColorImageFormat.RgbResolution640x480Fps30);
               newSensor.DepthStream.Enable(
                     DepthImageFormat.Resolution320x240Fps30);
               try
               {
                   // This will throw on non Kinect For Windows
                     devices.
                   newSensor.DepthStream.Range = DepthRange.Near;
                   newSensor.SkeletonStream.
                           EnableTrackingInNearRange = true;
               }
               catch (InvalidOperationException)
               {
                   [...]
               }

               newSensor.SkeletonStream.TrackingMode =
                                   SkeletonTrackingMode.Seated;
               newSensor.SkeletonStream.Enable();
               newSensor.AllFramesReady +=
                                   KinectSensorOnAllFramesReady;
           }
           [...]
       }
}
```

Así pues, el método SensorChooserOnKinectChanged() obtiene los argumentos del evento, de los cuales se pueden extraer referencias al antiguo sensor que se poseía y al nuevo sensor que se ha obtenido. Si el antiguo sensor sigue activo, lo que se hace es deshabilitar todas las funciones que se habían habilitado, para no interferir con las necesidades de otras aplicaciones que lo puedan estar usando. En el nuevo sensor, se activa el flujo de la cámara de color, el flujo de datos de profundidad de la cámara infrarroja, se activa el seguimiento de esqueletos y el flujo correspondiente de datos esqueletales. Por último, se indica que se quiere recibir eventos de tipo "AllFramesReady", los cuales serán procesados por el método KinectSensorOnAllFramesReady().

En el código se incluye una sección try-catch que, como bien se indica en el comentario, es capaz de detectar si se está usando un dispositivo Kinect para Windows, ya que es el único que soporta el Modo Cercano. Si no se está utilizando este dispositivo, se pasará a la sección catch, que activará el modo por defecto para que esta aplicación pueda usarse también con dispositivos Kinect de Xbox 360.

Para resumir, este método consigue lo siguiente:

1. Cortar todo lo que ata esta aplicación al dispositivo Kinect que se usaba anteriormente.
2. Activar el flujo de cámara a color, datos de profundidad y datos esqueletales en el nuevo dispositivo Kinect.
3. Obtener una notificación cada vez que el nuevo dispositivo Kinect tiene listos todos los fotogramas de los flujos activados.

Al igual que en el caso del constructor, el flujo de ejecución acaba aquí y el programa vuelve a entrar en modo de espera hasta recibir el evento "AllFramesReady". A partir de ahora, en un caso normal, la ejecución se limitará a esperar de nuevo este evento una vez se haya terminado de procesar el anterior, ya que no hay más funcionalidad. La ejecución se terminará cuando se cierre la ventana y esto invoque al método WindowClosed(), que se encarga de parar la ejecución de el resto de las clases del programa.

```
private void KinectSensorOnAllFramesReady(object sender,
AllFramesReadyEventArgs allFramesReadyEventArgs)
{
       using (var colorImageFrame =
                     allFramesReadyEventArgs.OpenColorImageFrame())
       {
           if (colorImageFrame == null)
           {
               return;
           }

           // Make a copy of the color frame for displaying.
           var haveNewFormat = this.currentColorImageFormat !=
                                        colorImageFrame.Format;
           if (haveNewFormat)
           {
               this.currentColorImageFormat =
                                        colorImageFrame.Format;
               this.colorImageData = new
                           byte[colorImageFrame.PixelDataLength];
               this.colorImageWritableBitmap = new WriteableBitmap(
                   colorImageFrame.Width, colorImageFrame.Height,
                   96, 96, PixelFormats.Bgr32, null);
               ColorImage.Source = this.colorImageWritableBitmap;

           }
```

```
            colorImageFrame.CopyPixelDataTo(this.colorImageData);
            this.colorImageWritableBitmap.WritePixels(
                new Int32Rect(0, 0, colorImageFrame.Width,
                colorImageFrame.Height), this.colorImageData,
                colorImageFrame.Width * Bgr32BytesPerPixel, 0);
        }
}
```

El manejo que le da el método KinectSensorOnAllFramesReady() al evento "AllFramesReady" es bastante sencillo. En los argumentos se encuentran todos los fotogramas generados, pero sólo se obtiene el contenido del fotograma de la cámara de color. Si no hay contenido (no se obtuvieron datos de la cámara a tiempo), el método no puede hacer nada. Pero si, por el contrario, sí que se ha obtenido el fotograma, lo que se hace es copiar la imagen al objeto "ColorImage" que se había declarado en la interfaz de usuario.

La forma de conseguir acceder a este objeto se encuentra en la segunda sección if del método, que además de adaptar el formato y la reserva de memoria al fotograma que se obtenga de la cámara, también vincula el objeto "ColorImage" a la copia del fotograma. De este modo, siempre que se cambien los datos almacenados en esa memoria, la imagen de la interfaz de usuario se actualizará. Este proceso se repite con cada nuevo fotograma hasta que se cierre la interfaz de usuario.

FaceTrackingViewer.xaml

Este archivo está esencialmente vacío; lo único que se hace es vincularlo a su clase de procesamiento interno. Esto es deseable, sin embargo, ya que la interfaz sigue existiendo, pero es transparente. El objetivo de esto es crear una capa transparente sobre la imagen de la cámara de color que se muestra en la interfaz de la ventana principal. En esta capa transparente se dibujará la malla de puntos faciales, con lo que se superpondrá sobre la imagen de vídeo.

FaceTrackingViewer.xaml.cs

Este archivo contiene dos clases distintas; FaceTrackingViewer y SkeletonFaceTracker. La primera asigna instancias de la segunda a cada esqueleto que Kinect reconozca, de modo que se puede entender que la segunda clase realiza tareas más específicas que la primera, cuya función es más general.

La clase FaceTrackingViewer es instanciada por la interfaz de usuario de la ventana principal. Su constructor contiene una única línea que inicia la interfaz de usuario (FaceTrackingViewer.xaml), no se producen otras acciones, de modo que la clase queda a la espera de que le notifiquen el único evento al que está suscrita; KinectChanged. Cuando esto se produce, el evento se procesa de una forma especial que acaba extrayendo dos de sus argumentos y enviándoselos al método OnSensorChanged().

```
private void OnSensorChanged(KinectSensor oldSensor, KinectSensor
newSensor)
{
       if (oldSensor != null)
       {
           oldSensor.AllFramesReady -= this.OnAllFramesReady;
           this.ResetFaceTracking();
       }

       if (newSensor != null)
       {
           newSensor.AllFramesReady += this.OnAllFramesReady;
       }
}
```

Este método, como se puede imaginar, hace algo parecido a lo que se hacía al procesar el mismo evento en la clase MainWindow; se cortan todas las referencias con el dispositivo Kinect anterior y se ajusta el nuevo. En este caso, para cortar dichas referencias, se deja de pedir notificaciones al dispositivo y se eliminan todas las instancias de SkeletonFaceTracker que se hayan creado bajo ese sensor. Esto último lo realiza con facilidad el método ResetFaceTracking(), que no merece la pena analizar a fondo, pues son dos líneas de código.

Tras cortar los enlaces, la clase se suscribe a los eventos de tipo AllFramesReady del dispositivo Kinect nuevo, que las procesará el método OnAllFramesReady(). Con esto, el flujo de ejecución del programa se vuelve a interrumpir hasta que se produzcan dichos eventos.

```
private void OnAllFramesReady(object sender, AllFramesReadyEventArgs
allFramesReadyEventArgs)
{
        [...]
        try
        {
            colorImageFrame =
                      allFramesReadyEventArgs.OpenColorImageFrame();
            depthImageFrame =
                      allFramesReadyEventArgs.OpenDepthImageFrame();
            skeletonFrame =
                      allFramesReadyEventArgs.OpenSkeletonFrame();

            [...]
            // Check for image format changes.  The FaceTracker
            // doesn't deal with that so we need to reset.
            [...]

            // Create any buffers to store copies of the data we work with
            [...]

            // Get the skeleton information
            [...]

            colorImageFrame.CopyPixelDataTo(this.colorImage);
            depthImageFrame.CopyPixelDataTo(this.depthImage);
            skeletonFrame.CopySkeletonDataTo(this.skeletonData);

            // Update the list of trackers and the trackers with the
               current frame information
            foreach (Skeleton skeleton in this.skeletonData)
            {
                if (skeleton.TrackingState == SkeletonTrackingState.Tracked
        || skeleton.TrackingState == SkeletonTrackingState.PositionOnly)
                {
                    // We want keep a record of any skeleton, tracked or
                      untracked.
                    if (!this.trackedSkeletons.ContainsKey
                                                (skeleton.TrackingId))
                    {
                        this.trackedSkeletons.Add(skeleton.TrackingId,
                                            new SkeletonFaceTracker());
                    }

                    // Give each tracker the upated frame.
                    SkeletonFaceTracker skeletonFaceTracker;
                    if (this.trackedSkeletons.TryGetValue
                      (skeleton.TrackingId, out skeletonFaceTracker))
                    {
                        skeletonFaceTracker.OnFrameReady(this.Kinect,
                             colorImageFormat, colorImage,
                             depthImageFormat, depthImage, skeleton);
                        skeletonFaceTracker.LastTrackedFrame =
                                            skeletonFrame.FrameNumber;
                    }
                }
            }
            this.RemoveOldTrackers(skeletonFrame.FrameNumber);
            this.InvalidateVisual();
        }
        finally
        [...]
}
```

Este método es el más complejo de toda la clase FaceTrackingViewer, pero se puede comprender su funcionamiento si uno se fija bien en lo que va haciendo, sin dejarse distraer por las comprobaciones y declaraciones adicionales. Lo primero es extraer los datos interesantes de los argumentos, que vienen a ser todos. Una vez extraídos, se comprueba que contienen datos, puesto que si falta alguno de los fotogramas no se puede realizar el seguimiento facial. Si están todos, se comprueba que el formato de los fotogramas se corresponda con el que se tiene. Si no (si el dispositivo Kinect ha cambiado), se tendrán que renovar los búferes de almacenamiento para adaptarlos a las nuevas dimensiones. Lo mismo pasa con el array que contiene los esqueletos.

Una vez hecho esto, se copian los datos a los búferes. Ya se tienen todos los datos con los que se van a trabajar. Ahora, el método pasa a comprobar que todos los esqueletos que se han detectado en el fotograma tengan una instancia de SkeletonFaceTracker asignada. Si no, se crea una nueva instancia de esa clase y se le asigna al esqueleto.

Ahora que todos los esqueletos tienen una instancia de la clase, se llama al método OnFrameReady() de dicha clase pasándole los fotogramas actuales. De este modo, cada instancia los procesará y tendrá lista la malla de puntos faciales de cada esqueleto. También se les actualiza a las instancias el número de fotograma actual.

Una vez se terminan las llamadas, se comprueba qué instancias llevan mucho sin actualizar su número de fotograma y se eliminan, ya que probablemente la persona se haya ido y no vuelva. El paso siguiente es invalidar lo que se muestra en la interfaz de usuario de FaceTrackingViewer, para forzar un nuevo renderizado y mostrar los datos nuevos. Esta llamada se explicará más adelante.

En resumen, con este método se logra lo siguiente:

1. Extraer los fotogramas de los flujos activos. Se eliminan al final, independientemente de si se usan o no.
2. Asignar a cada esqueleto una instancia de SkeletonFaceTracker, que se encargará de obtener la malla facial de dicho individuo.
3. Proveer los datos de fotograma nuevos a cada una de esas instancias.
4. Eliminar las instancias viejas cuyos esqueletos no aparezcan desde hace tiempo.
5. Provocar que se actualice la interfaz de usuario.

El flujo de ejecución se desvía tres veces durante este método; para crear las instancias de SkeletonFaceTracker, para obtener las mallas faciales de cada uno de ellos, y para invalidar la interfaz de usuario. La clase SkeletonFaceTracker no posee contructor, de modo que primero se explicará el contenido del método OnFrameReady() que se utilizó en segundo lugar. Al fin y al cabo, va delante en el orden del flujo de ejecución.

```
internal void OnFrameReady(KinectSensor kinectSensor,
ColorImageFormat colorImageFormat, byte[] colorImage,
DepthImageFormat depthImageFormat, short[] depthImage, Skeleton
skeletonOfInterest)
{
       this.skeletonTrackingState =
                            skeletonOfInterest.TrackingState;
       [...]

       if (this.faceTracker == null)
       {
           try
           {
               this.faceTracker = new FaceTracker(kinectSensor);
           }
           catch (InvalidOperationException)
           {
               [...]
           }
       }

       if (this.faceTracker != null)
       {
           FaceTrackFrame frame = this.faceTracker.Track(
                   colorImageFormat, colorImage, depthImageFormat,
                   depthImage, skeletonOfInterest);

           this.lastFaceTrackSucceeded = frame.TrackSuccessful;
           if (this.lastFaceTrackSucceeded)
           {
               [...]

               this.facePoints = frame.GetProjected3DShape();
           }
       }
}
```

En primer lugar, este método comprueba si el esqueleto que tiene asignado está bajo seguimiento activo. Si no es así, no tiene sentido intentar obtener una malla de puntos faciales. Una vez se sabe que está bajo seguimiento activo, se comprueba si se tiene una instancia de la clase FaceTracker (del paquete de Toolkit.FaceTracking), que se necesitará para el seguimiento facial. Si no se tiene, se crea.

Ahora que se tiene la instancia de FaceTracker, se le pide obtenga toda la información facial del esqueleto/individuo actual por medio del método Track(), y se comprueba si se ha conseguido obtener dicha información. Si así ha sido, se almacena la estructura triangular de la malla facial si no se tiene, y se guardan los puntos faciales en dos dimensiones. Se hace así para que se puedan mostrar directamente sobre la imagen de la cámara de color sin que parezcan descolocados.

Una vez se acaba la ejecución, el flujo vuelve al método OnAllFramesReady() de la clase FaceTrackingViewer, ya que es el único que llama a este otro método. Después de continuar con la ejecución de OnAllFramesReady() de la forma indicada anteriormente, se llega a la llamada al método InvalidateVisual(). Este método es propio de las clases que controlan el proceso interno de una interfaz, ya que les permite indicar a la interfaz que lo que se muestra está desfasado y debe eliminarse. Tras la ejecución de este método, que es interno y no se puede ver el código fuente, se garantiza que siempre se va a llamar al método OnRender(). Este método se encuentra en la clase FaceTrackingViewer, como es lógico.

```
protected override void OnRender(DrawingContext drawingContext)
{
        base.OnRender(drawingContext);
        foreach (SkeletonFaceTracker faceInformation in
                                    this.trackedSkeletons.Values)
        {
            faceInformation.DrawFaceModel(drawingContext);
        }
}
```

Este método es simple. Lo primero es invocar el mismo método en la interfaz de usuario propia (FaceTrackingViewer.xaml) para que vuelva a mostrar los elementos estáticos. Luego se recorre todas las instancias de la clase SkeletonFaceTracker y llama al método DrawFaceModel() de cada una de ellas. De este modo, cada instancia dibuja su malla de puntos faciales, si es que se consiguieron los datos durante la ejecución de su método OnFrameready().

```
public void DrawFaceModel(DrawingContext drawingContext)
{
        [...]
        var faceModelPts = new List<Point>();
        var faceModel = new List<FaceModelTriangle>();

        for (int i = 0; i < this.facePoints.Count; i++)
        {
            faceModelPts.Add(new Point(this.facePoints[i].X + 0.5f,
                                    this.facePoints[i].Y + 0.5f));
        }

        foreach (var t in faceTriangles)
        {
            var triangle = new FaceModelTriangle();
            triangle.P1 = faceModelPts[t.First];
            triangle.P2 = faceModelPts[t.Second];
            triangle.P3 = faceModelPts[t.Third];
            faceModel.Add(triangle);
        }
```

```
        var faceModelGroup = new GeometryGroup();
        for (int i = 0; i < faceModel.Count; i++)
        {
            var faceTriangle = new GeometryGroup();
            faceTriangle.Children.Add(new
               LineGeometry(faceModel[i].P1, faceModel[i].P2));
            faceTriangle.Children.Add(new
               LineGeometry(faceModel[i].P2, faceModel[i].P3));
            faceTriangle.Children.Add(new
               LineGeometry(faceModel[i].P3, faceModel[i].P1));
            faceModelGroup.Children.Add(faceTriangle);
        }

        drawingContext.DrawGeometry(Brushes.LightYellow, new
               Pen(Brushes.LightYellow, 1.0), faceModelGroup);
}
```

Este método, como todo método que implica añadir algo visible a una interfaz de usuario, realiza muchas operaciones para obtener un resultado relativamente pequeño. Se comienza comprobando que se obtuvieron correctamente los datos faciales del fotograma actual, y si así fue, se añaden los puntos faciales obtenidos a una nueva lista. Se introduce un ligero cambio en las coordenadas para compensar por el grosor de las líneas que se van a dibujar. Así no aparentarán estar ligeramente desviadas.

Tras esto, se organizan los puntos en triángulos, gracias a la estructura de la malla facial que se había guardado anteriormente. Así pues, siguiendo el mismo orden que en esa estructura, se crean triángulos con los puntos y se almacenan en una nueva lista. Esta lista se recorre a su vez para crear instancias de LineGeometry, que viene a ser la representación gráfica de una línea, para cada uno de los lados del triángulo. Todas estas instancias se almacenan para ser finalmente dibujadas con la última instrucción del método. Esto las hace visibles en la interfaz de usuario, donde aparecen superpuestas sobre las caras de la imagen de la cámara a color.

Al terminar todas las llamadas a este método, la ejecución pasa a modo de espera hasta que se produzca el siguiente evento o se cierre la ventana de la interfaz de usuario, con lo que simplemente se borran todas las instancias de ambas clases, FaceTrackingViewer y SkeletonFaceTracker.

Ese era el último método que faltaba por analizar, de modo que ese era todo el análisis del código fuente de la aplicación de ejemplo. Como se puede observar, contiene una serie de métodos útiles que se podrían reutilizar, reciclar o ampliar en futuros proyectos, y eso es lo que se ha hecho con el presente proyecto.

Sistema de "escaparate caliente"

En cualquier comercio se conoce la importancia de anunciar los productos que se ofrecen para aumentar la demanda de los mismos. Si un producto se oferta de la forma apropiada, se puede inducir a la compra a los clientes potenciales. Por esta razón se crearon los anuncios en los diferentes medios; televisión, radio, carteles fijos, sponsors, etc. Pero quizá la forma de publicidad más importante de todas consiste en el escaparate[7].

El escaparate es la fachada de cualquier comercio, la cara exterior, la primera impresión. Un escaparate atractivo no sólo muestra algunos productos, sino que incita a los clientes potenciales a entrar en la tienda. Es por ello que siempre se muestra especial cuidado en la distribución y en los productos que se muestran en dicho escaparate; pone al público en una situación de deseo[7] y puede significar la diferencia entre un éxito de ventas y una recaudación mediocre.

El sistema de "escaparate caliente" surge para resolver el problema de saber si los productos exhibidos son populares entre la gente que se para a mirar. Es decir, permite saber a qué lugares del escaparate se ha mirado más y a cuáles menos, obteniendo así información sobre los productos y los objetos decorativos que llaman la atención y los que no. Esta información puede ser de utilidad a la hora de saber si se debe cambiar la presentación de un escaparate, qué cosas conviene cambiar y qué cosas es mejor que permanezcan por su popularidad.

Así pues, los objetivos del sistema de "escaparate caliente" son:

- Reconocer a dónde está mirando una persona que se encuentre examinando el escaparate.
- Distinguir los lugares que interesan a la persona en contraste con los que sólo mira por encima.
- Si es posible, poder recabar los datos de observación de más de una persona a la vez, puesto que en ocasiones hay más de una persona mirando al escaparate.
- Si es posible, obtener datos de la persona en sí, ya sea edad, género, u otra información que pueda resultar útil.
- Hacer que los datos sean accesibles para los administradores del comercio. Es decir, que los datos se puedan comprender y estudiar con facilidad sin necesidad de conocimientos de informática.

Los objetivos del sistema son claros, pero los problemas que plantea no son todos tan evidentes. En primer lugar, el proceso de calcular a dónde está mirando una persona es costoso, ya que la forma normal de realizar este cálculo es sobre una imagen en dos dimensiones que proporciona una cámara. Si este proceso es pesado con una sola persona, que el sistema sea capaz de reconocer dos o más es un problema aún mayor, aunque tiene más que ver con la optimización del proceso de cálculo y la eficiencia del hardware usado.

Otro problema a resolver, si bien parece opcional según los objetivos del sistema, consiste en qué datos obtener del cliente potencial y cómo obtenerlos. La mayoría de las respuestas a la segunda pregunta acaban necesitando o bien alguna forma de visión artificial o algún tipo de aprendizaje automático. Cuanto más se sabe de ese cliente potencial, más fácil es mostrar un producto que pueda interesarle. Por eso este problema es uno de los más importantes a resolver.

La forma de hacer que los datos sean comprensibles para los administradores del comercio es un problema en sí mismo. No sólo se trata de la forma de representar los datos, sino de la forma de almacenarlos y recuperarlos.

La prueba de concepto

Para realizar la prueba de concepto, se pretende realizar un prototipo del sistema, es decir, una implementación que no incluya todas las características que se esperan de una aplicación final. Se va a obviar el proceso final de optimización y se utilizarán formas básicas de obtención de información del potencial cliente y de representación de la información.

Si la implementación parcial obtiene buenos resultados, será un indicador de que el sistema es viable y se puede considerar su construcción completa y su uso en entornos comerciales reales.

Arquitectura

La arquitectura escogida sigue a grandes rasgos la arquitectura que tenía el código de ejemplo que se ha usado como base. El sistema se divide en tres capas, donde cada capa sólo puede comunicarse con la anterior o la siguiente. En primer lugar está la Capa de Usuario, que es la capa con la que puede interactuar el usuario del sistema. Esta capa incluye la Interfaz de usuario y los Archivos. La siguiente capa es la Capa de Software, a la que pertenece el resto de la aplicación, incluido el SDK de Kinect. La última capa es la Capa de Dispositivos, que es donde se encuentran todos los dispositivos Kinect que se vayan a usar, si bien la aplicación está diseñada para funcionar sólo con uno.

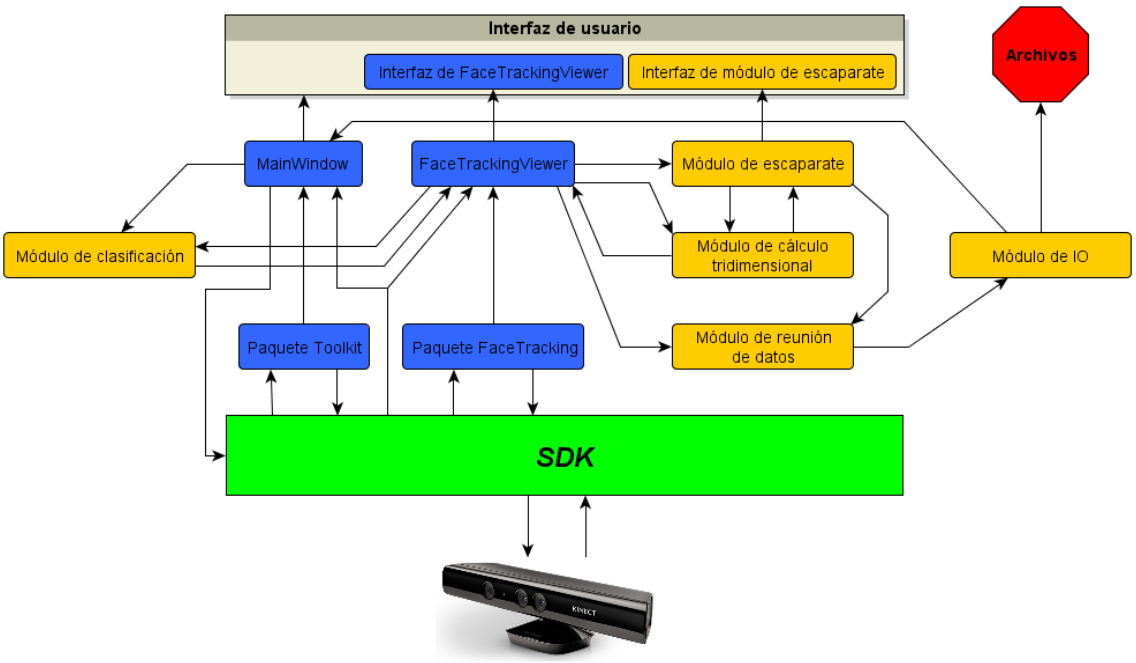


Figura 9: *Arquitectura de la aplicación desarrollada para la prueba de concepto.*

Como se puede apreciar en la Figura 9, el sentido de la información es principalmente ascendente en esta arquitectura, ya que desde que se inicia la aplicación, lo único que se hace es generar información desde la Capa de Dispositivos, que se acabará mostrando al usuario por medio de la Interfaz de Usuario o los Archivos.

El sistema diseñado es ampliamente modular, ya que cada módulo realiza una tarea concreta y se comunica con los demás para obtener o enviar la información que se necesite. La razón por la que se ha adoptado este diseño es simple; es la mejor forma de añadir nueva funcionalidad a una aplicación ya existente. Además, también cuenta con la ventaja de que cada módulo es independiente de los demás, con lo que al corregir un error normalmente sólo se debe modificar uno de ellos, por ejemplo.

De lo que muestra la Figura 9, los módulos en azul son los que pertenecen al código de ejemplo, los que aparecen en amarillo son los nuevos módulos que se han implementado para crear la aplicación, mientras que el SDK de Kinect se muestra en verde y la Interfaz de usuario tiene un color gris. La figura en color rojo se refiere a archivos presentes en el sistema local de almacenamiento del ordenador.

A pesar de la aparente complejidad del flujo de información, en realidad el proceso es asequible y se puede comprender con un poco de esfuerzo. Si se eliminan los módulos en amarillo, las flechas restantes se describen en el Análisis del código de ejemplo, la sección anterior de la presente memoria. Una vez se sabe qué hace cada una simplemente se pueden añadir módulos uno a uno, con sus respectivas flechas, y

poco a poco ver las nuevas interacciones que se producen. De este modo se puede llegar a comprender el funcionamiento del programa.

Módulos detallados

En este apartado se detallarán los módulos más importantes de la aplicación. Esta sección se complementa con la sección anterior de Análisis del código de ejemplo, ya que la aplicación se ha desarrollado con la base proporcionada por dicho código. Sin embargo, hay que tener en cuenta que varias de las partes del código de ejemplo han sufrido modificaciones para adaptarse a la nueva arquitectura, de modo que es muy probable que las que se han incluido en la aplicación difieran en gran medida de las analizadas en el código de ejemplo.

Módulo de escaparate

El módulo de escaparate se encarga de recibir los vectores de mirada que se han calculado en el módulo FaceTrackingViewer y de realizar los cálculos necesarios para obtener su punto de intersección con el escaparate. Además de esto, debe ser capaz de manejar los datos de varios puntos a la vez y comprobar si alguno de ellos pasa al estado de mirada fija y cuánto tiempo (real) permanece en dicho estado. Todos los puntos de intersección encontrados se enviarán a la interfaz dependiente de este módulo para su representación en la Interfaz de usuario. Los datos de cada mirada fija que se produzca se envían al módulo de reunión de datos.

Módulo de cálculo tridimensional

Uno de los mayores problemas que se tienen que resolver en este sistema es el manejo de datos en tres dimensiones. El vector de la mirada de una persona, los datos de su esqueleto y las coordenadas del escaparate son algunas de las piezas de información que se usan en el espacio tridimensional. Por fortuna, todos ellos se obtienen con respecto al origen de coordenadas que fija el dispositivo Kinect, de modo que no hay que efectuar transformaciones de coordenadas.

Así pues, este módulo se encarga de los cálculos que se deban hacer sobre algún dato en tres dimensiones. Principalmente, se trata del cálculo del vector de mirada a partir de la malla de puntos faciales de una persona y el cálculo de la intersección entre la dirección de la mirada y el plano del escaparate. Pero también se tienen que realizar proyecciones para convertir datos en tres dimensiones a dos dimensiones.

Esto último se debe al hecho de que el SDK del dispositivo Kinect puede devolver puntos del esqueleto y de la cara ya proyectados, pero no se le puede pedir que proyecte unos puntos que se le indiquen. Así pues, se ha tenido que desarrollar un método de proyección de 3D a 2D de forma que la imagen proyectada coincida con los fotogramas de la cámara de color y se pueda superponer a los mismos. La proyección del punto de intersección de la mirada con el escaparate es mucho más sencilla en comparación.

```
Ubicación de archivos de salida:: Archivos\Datos prueba\
Ubicación de red de neuronas entrenada:: Archivos\Red Neuronal, e 000-058, v 059-065.txt
Dimensiones del escaparate respecto a Kinect (sólo el cristal)::
 - Distancia a escaparate:: 0.55 m
 - Distancia al lado izquierdo del escaparate:: 0.44 m
 - Distancia al lado derecho del escaparate:: 1.69 m
 - Distancia al lado inferior del escaparate:: 0.24 m
 - Altura total del escaparate:: 0.69 m

Opciones::
 - Dibujo de los esqueletos en la imagen:: ON
 - Kinect en modo cercano:: OFF
 - Guardar esqueletos en lugar de datos de salida normales:: ON
 - Generar una red de neuronas entrenada en lugar del funcionamiento estándar:: OFF

Datos internos:: 000 058 059 065 351

```

Figura 10: *Ejemplo de archivo de configuración.*

Módulo de reunión de datos

Este módulo se encarga de recibir y organizar los datos de cada persona sobre la que el seguimiento facial ha tenido éxito. Así pues, por cada una de estas personas, debe almacenar su esqueleto, el tipo de persona que se ha reconocido y, de haberlos, los puntos del escaparate que han sido de interés para ella.

Esta información se mantiene siempre que la persona permanezca en el alcance de la cámara de infrarrojos, aunque no se encuentra bajo seguimiento activo de esqueleto. Se debe recordar que el dispositivo Kinect puede hacer seguimiento esqueletal activo de dos personas e inactivo de otras cuatro. Una vez se haya perdido a la persona, y siempre que no vuelva antes de que transcurra un cierto tiempo, se organizan los datos y se mandan al Módulo de IO.

Módulo de IO

IO, o I/O como se escribe normalmente, es un término que toma las iniciales de "Input/Output", Entrada/Salida en inglés. Así pues, este módulo se encarga de las operaciones de entrada y salida de datos relacionados con ficheros en el sistema local de archivos. O dicho de otro modo, se encarga de leer y escribir datos en archivos.

```
Id Personal: 625
Categoría: Mujer
Lugares a los que ha mirado un tiempo:
 (1,9230; 0,6376)   durante 6,4 segundos
 (0,7915; 0,2235)   durante 2,36 segundos
 (1,0635; 0,3324)   durante 2,4 segundos
 (1,1713; 0,3616)   durante 2,64 segundos
 (1,5851; 0,6078)   durante 2,1 segundos
 (1,6996; 0,5891)   durante 2,35 segundos
 (1,6276; 0,4405)   durante 8,67 segundos

```

Figura 11: *Ejemplo de archivo de puntos de interés.*

Este módulo tiene tres funciones principales; leer el archivo de configuración (Figura 10) al inicio de la ejecución, cargar la red de neuronas artificiales que se especifique en dicho archivo y escribir los datos que proporcione el módulo de reunión de datos en archivos individuales para cada persona detectada. Se pueden encontrar ejemplos de los archivos que se producen en las Figuras 11 y 12.

Módulo de clasificación

El módulo de clasificación es el que permite discernir a qué tipo pertenece una persona a partir de los datos que se han obtenido sobre ella. En una implementación real de este sistema, probablemente se obtendrán una serie de datos útiles sobre el potencial cliente, pero para esta prueba de concepto, se ha decidido que con clasificar a las personas entre los tipos "Hombre", "Mujer" y "Niño" será suficiente.

Así pues, para realizar esta clasificación, el módulo cuenta con una implementación de una red de neuronas multicapa entrenable, cuyos datos se pueden exportar a un archivo de texto para poder usarlos en sucesivas ejecuciones. En el archivo de configuración se indica la ubicación de los datos de la red de neuronas que se va a utilizar en la ejecución.

```
1  Head_To_Shoulder_Center: 0,173081794775683
2  Shoulder_Center_To_Right:0,213457737153133
3  Shoulder_To_Elbow_Right: 0,205707172940133
4  Elbow_To_Wrist_Right:    0,197017635331999
5  Wrist_To_Hand_Right:     0,0649705258095881
6  Shoulder_Center_To_Left: 0,173060720337489
7  Shoulder_To_Elbow_Left:  0,230490149275982
8  Elbow_To_Wrist_Left:     0,206052384264933
9  Wrist_To_Hand_Left:  0,0591347139410681
10 Shoulder_Center_To_Spine:0,328988224654816
11 Spine_To_Hip_Center:     0,0738904296727826
12 Hip_Center_To_Right:     0,0959078345547182
13 Hip_To_Knee_Right:   0,298892060414189
14 Knee_To_Ankle_Right:     0,297809991940634
15 Ankle_To_Foot_Right:     0,0867790207138757
16 Hip_Center_To_Left:  0,0909757565220538
17 Hip_To_Knee_Left:    0,321385400049252
18 Knee_To_Ankle_Left:  0,266043950321399
19 Ankle_To_Foot_Left:  0,0831249567494078
20 Tipo: Mujer
21
```

Figura 12: *Ejemplo de archivo de esqueleto.*

La razón por la que se ha decidido usar aprendizaje automático para obtener datos de los potenciales clientes es que se trata del mejor sistema para el entorno propuesto. El entorno es totalmente abierto, no tiene restricciones ni está parametrizado. Los únicos datos que se tienen son dos imágenes y las coordenadas de un esqueleto en tres dimensiones. Ya el propio dispositivo Kinect utiliza el aprendizaje automático para obtener dicho esqueleto, así que el paso lógico es ir más allá y utilizar el aprendizaje automático para distinguir el tipo al que pertenece una persona.

El módulo de clasificación entra en funcionamiento cuando el módulo de FaceTrackingViewer consigue realizar el seguimiento facial de una persona. Dicho módulo proporciona los datos necesarios al módulo de clasificación para que determine a qué tipo pertenece dicha persona. Más adelante, el módulo de FaceTrackingViewer le enviará los datos de la clasificación al módulo de reunión de datos para su organización.

Pruebas de verificación y validación

Las pruebas son las que van a indicar tanto si se está construyendo el sistema correctamente como la viabilidad de la prueba de concepto una vez se haya terminado de implementar. Las pruebas de verificación se encargan de comprobar lo primero, mientras que son las pruebas de validación las que justifican lo segundo.

Pruebas de verificación

Las pruebas de verificación se llevan a cabo con uno o varios sujetos de prueba y con formas de iluminación variables a lo largo del desarrollo e implementación del sistema. Debido a la variación en la disponibilidad de dichos sujetos de prueba, hay escenarios que se consideraron en el momento de diseño de las pruebas que no se han llegado a poder probar en los momentos previstos.

Sin embargo, se considera que en caso de haberse podido probar en un momento posterior, si en esa prueba se obtuvieron resultados positivos, en la prueba perdida se habrían obtenido los mismos resultados o peores. Esto se debe a que el desarrollo del sistema es incremental y una vez que las pruebas de una parte del sistema dan resultados positivos, esa parte permanece inalterada a no ser que se especifique lo contrario, y se construye sobre ella.

Se ha considerado que se realice una batería de pruebas de verificación cuando la implementación alcance los siguientes hitos:

1. Se consigue obtener el vector de dirección de la mirada de una persona.
2. Se consigue calcular la posición del escaparate a la que mira una persona.
3. Se consigue determinar cuándo una persona se ha quedado mirando una misma zona por un espacio de tiempo.
4. Se consigue distinguir el tipo de persona (según lo especificado en el diseño) de un observador.

5. Se consigue registrar todos los datos de una sesión; tanto los esqueletos de los observadores como los puntos que les interesaron.

A cada hito se le asignan sus propias pruebas, ya que la implementación cada vez cuenta con más funcionalidades que se han de probar. Aunque, como se ha dicho antes, se asume que una funcionalidad que se probó antes seguirá funcionando correctamente en el siguiente hito. De no ser así, las siguientes pruebas mostrarían errores y se podría identificar el fallo que se haya producido.

Una vez se ha superado la batería de pruebas, se debe probar la implementación exitosa con dos sujetos de prueba a la vez, y comprobar que ambos obtienen los mismos resultados en las pruebas de la batería. Es recomendable usar incluso un tercer sujeto de prueba, para comprobar que no interfiere con ninguna de las funcionalidades probadas.

Primer hito

La batería de pruebas para el primer hito es la siguiente:

1. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. El sujeto mueve los brazos y la cabeza hasta que se muestre la malla facial. Se anota el resultado:
    a. El vector de la mirada no aparece.
    b. El vector de la mirada aparece, pero está muy desviado con respecto a donde está mirando el sujeto.
    c. El vector de la mirada aparece, pero en sentido contrario, apuntando hacia el interior de la cabeza.
    d. El vector de la mirada aparece, dirigido aproximadamente a donde está mirando el sujeto.
2. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Una vez se encuentra bajo seguimiento facial, el sujeto mueve la cabeza en pequeños círculos, de forma que no se pierda el seguimiento facial en ningún momento. Se anota el resultado:
    a. El vector de la mirada no aparece.
    b. El vector de la mirada aparece, pero se queda fijo en la posición inicial. (Independientemente de si, a pesar de estar fijo, cambia su dirección.)
    c. El vector de la mirada aparece y permanece en el punto original de la malla facial, pero su dirección no cambia.
    d. El vector de la mirada aparece, permanece en su punto de la malla facial a lo largo de su movimiento y su dirección cambia cuando lo hace la dirección de la mirada del sujeto de prueba.
    e. El vector de la mirada aparece, pero en algún momento de la prueba desaparece o su sentido se invierte.

3. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Una vez se encuentra bajo seguimiento facial, el sujeto acerca la cara hacia el dispositivo hasta que se pierda el seguimiento facial. Se anota el resultado:
    a. El vector de la mirada no aparece.
    b. El vector de la mirada aparece, pero desaparece antes de que se pierda el seguimiento facial y no vuelve a aparecer mientras se conserva este seguimiento. (Independientemente de si se queda fijo en un lugar o una dirección.)
    c. El vector de la mirada aparece, pero no desaparece incluso después de haber perdido el seguimiento facial.
    d. El vector de la mirada aparece, y desaparece a la vez que se pierde el seguimiento facial.
    e. El vector de la mirada aparece, y desaparece a la vez que se pierde el seguimiento facial, pero su longitud cambia al acercarse a la cámara del dispositivo.
    f. El vector de la mirada aparece, y desaparece a la vez que se pierde el seguimiento facial, pero su sentido se invierte en algún momento mientras el sujeto se acerca al dispositivo.

Si en esta batería de pruebas no se obtiene un resultado de "d", significa que la prueba ha sido fallida y se debe revisar la implementación. Repetir el proceso de prueba, análisis del error y reparación del error hasta que se obtenga el resultado "d" en las tres pruebas. Los resultados de las pruebas realizadas se recogen en la Tabla 4.

Implementación	Resultado Pr. 1	Resultado Pr. 2	Resultado Pr. 3
1ª	d		b
2ª	c		
3ª	d	d	d
4ª (puntos alternativos para obtener el vector)	c		
5ª	d	d	d
6ª (puntos alternativos para obtener el vector)	d	d	d
7ª (puntos alternativos para obtener el vector)	d	d	d
Nota adicional: Se ha probado correctamente la 7ª implementación con dos sujetos.			

Tabla 4: *Resultados de la primera batería de pruebas.*

Segundo hito

La batería de pruebas para el segundo hito es la siguiente. Se requiere el uso de un marco o algún tipo de indicador que represente el escaparate para que el sujeto sepa a qué superficie debe mirar.

1. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Marco colocado y sus coordenadas introducidas. El sujeto mira a cualquier punto dentro del marco. Se anota el resultado:
    a. No aparece un punto de intersección en la pantalla.
    b. En la pantalla aparece un punto de intersección, pero no está dentro del escaparate.
    c. En la pantalla aparece un punto de intersección dentro del escaparate.
2. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Marco colocado a metro y medio del sujeto, con sus coordenadas introducidas. El sujeto mira al centro del marco. Se anota el resultado:
    a. No aparece el punto de intersección en la pantalla.
    b. En la pantalla aparece un punto de intersección, pero no está dentro del escaparate.
    c. En la pantalla aparece un punto de intersección en el centro del escaparate o en un área cercana.
    d. En la pantalla aparece un punto de intersección, pero se desvía del centro considerablemente hacia uno de los lados.
    e. En la pantalla aparece un punto de intersección, pero se desvía del centro considerablemente hacia arriba o abajo.
3. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Marco colocado y sus coordenadas introducidas. El sujeto mira a una de las esquinas del marco y luego a la esquina contraria, haciendo un movimiento en diagonal. Se anota el resultado:
    a. No aparece el punto de intersección en la pantalla.
    b. En la pantalla aparece un punto de intersección, pero desaparece durante el desplazamiento, volviendo o no a aparecer después.
    c. En la pantalla aparece un punto de intersección, que se mueve en la misma dirección que la mirada del sujeto.
    d. En la pantalla aparece un punto de intersección que se mueve en la misma dirección que la mirada del sujeto, pero uno de los movimientos (horizontal o vertical) es claramente erróneo. Que sea la mitad de lo que debería, por ejemplo.
    e. En la pantalla aparece un punto de intersección que se mueve en la misma dirección que la mirada del sujeto, pero el movimiento

parece descuadrado. El punto empieza dentro del escaparate pero se sale antes de terminar el movimiento.

f. En la pantalla aparece un punto de intersección que se mueve en la misma dirección que la mirada del sujeto, pero en sentido contrario. Es decir, si el sujeto mira de la esquina superior derecha a la inferior izquierda, el punto se desplaza de la segunda a la primera.

g. En la pantalla aparece un punto de intersección, pero sólo uno de los movimientos es el contrario del que debería. Es decir, el punto se mueve de izquierda a derecha cuando el sujeto mira en diagonal de derecha a izquierda, pero el movimiento vertical es correcto.

Si en esta batería de pruebas no se obtiene un resultado de "c", significa que la prueba ha sido fallida y se debe revisar la implementación. Aunque si se trata de la prueba número dos, es probable que la implementación que se deba revisar sea la del módulo anterior, en concreto, los puntos utilizados para el cálculo del vector de mirada. Repetir el proceso de prueba, análisis del error y reparación del error hasta que se obtenga el resultado "c" en las tres pruebas. Los resultados de las pruebas realizadas se recogen en la Tabla 5.

Implementación	Resultado Pr. 1	Resultado Pr. 2	Resultado Pr. 3
1ª	c	e	
2ª	c	e	
3ª	c	e	
4ª	c	c	e
5ª	c		g
6ª	c	c	e
7ª	c	c	c
Nota adicional: Se ha probado correctamente la 7ª implementación con dos sujetos.			

Tabla 5: *Resultados de la segunda batería de pruebas.*

Tercer hito

La batería de pruebas para el tercer hito es la siguiente. También requiere el uso de un marco o indicador del espacio del escaparate.

1. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Marco colocado y sus coordenadas introducidas. El sujeto mira fijamente durante unos segundos al mismo punto. Se anota el resultado:
    a. El punto de intersección no cambia para indicar mirada fija.
    b. El punto de intersección cambia para indicar mirada fija.
    c. El punto de intersección indica mirada fija desde el principio.
    d. El punto de intersección cambia para indicar mirada fija, pero vuelve a cambiar al punto normal sin haberse movido de sitio.
2. Kinect con flujo esqueletal en Modo Sentado. Sujeto de prueba sentado o de pie de frente a la cámara, a una distancia suficiente para que se le vea al menos de cabeza a cintura. Marco colocado y sus coordenadas introducidas. El sujeto mira fijamente durante unos segundos a una zona pequeña, moviendo la cabeza ligeramente. Se anota el resultado:
    a. El punto de intersección no cambia para indicar mirada fija. (Comprobar que el movimiento de la mirada no es demasiado amplio.)
    b. El punto de intersección cambia para indicar mirada fija.

Si en esta batería de pruebas no se obtiene un resultado de "b", significa que la prueba ha sido fallida y se debe revisar la implementación. Repetir el proceso de prueba, análisis del error y reparación del error hasta que se obtenga el resultado "b" en las dos pruebas. Los resultados de las pruebas realizadas se recogen en la Tabla 6.

Implementación	Resultado Pr. 1	Resultado Pr. 2
1ª	b	b
Nota adicional: Se ha probado la 1ª implementación con dos sujetos, y para uno de ellos no se superaban las dos pruebas.		
2ª	b	b
Nota adicional: Se ha probado correctamente la 2ª implementación con dos sujetos.		

Tabla 6: *Resultados de la tercera batería de pruebas.*

Cuarto hito

La batería de pruebas para el cuarto hito es la siguiente:

1. Kinect con flujo esqueletal en Modo por Defecto. Sujeto de prueba de tipo "Hombre" de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. El sujeto mira en dirección al dispositivo Kinect de modo que se active el seguimiento facial. Se repite dos veces más con sujetos del mismo tipo y se anotan los resultados:
    a. No se reconoce el tipo de alguno de los sujetos.
    b. El tipo reconocido de todos los sujetos es "Hombre".
    c. El tipo reconocido de dos de los sujetos es "Hombre".
    d. El tipo reconocido de uno de los sujetos es "Hombre".
    e. No se reconoce a ningún sujeto como perteneciente al tipo "Hombre".
2. Kinect con flujo esqueletal en Modo por Defecto. Sujeto de prueba de tipo "Mujer" de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. El sujeto mira en dirección al dispositivo Kinect de modo que se active el seguimiento facial. Se repite dos veces más con sujetos del mismo tipo y se anotan los resultados:
    a. No se reconoce el tipo de alguno de los sujetos.
    b. El tipo reconocido de todos los sujetos es "Mujer".
    c. El tipo reconocido de dos de los sujetos es "Mujer".
    d. El tipo reconocido de uno de los sujetos es "Mujer".
    e. No se reconoce a ningún sujeto como perteneciente al tipo "Mujer".
3. Kinect con flujo esqueletal en Modo por Defecto. Sujeto de prueba de tipo "Niño" de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. El sujeto mira en dirección al dispositivo Kinect de modo que se active el seguimiento facial. Se repite una vez más con otro sujeto del mismo tipo y se anotan los resultados:
    a. No se reconoce el tipo de alguno de los sujetos.
    b. El tipo reconocido de todos los sujetos es "Niño".
    c. El tipo reconocido de uno de los sujetos es "Niño".
    d. No se reconoce a ningún sujeto como perteneciente al tipo "Niño".
4. Kinect con flujo esqueletal en Modo por Defecto. Sujeto de prueba de cualquier tipo de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. El sujeto mira hacia algún lado de modo que NO se active el seguimiento facial. Se anotan los resultados:
    a. No se reconoce el tipo del sujeto.
    b. El tipo reconocido del sujeto es "Indeterminado".
    c. El tipo reconocido del sujeto es otro distinto a "Indeterminado".

Si en esta batería de pruebas no se obtiene un resultado de "b", significa que la prueba ha sido fallida y se debe revisar la implementación. Repetir el proceso de prueba, análisis del error y reparación del error hasta que se obtenga el resultado "b" en las dos pruebas. Los resultados de las pruebas realizadas se recogen en la Tabla 7.

Implementación	Result. Pr. 1	Result. Pr. 2	Result. Pr. 3	Result. Pr. 4
1ª	b	c (con sólo dos sujetos)		a
2ª	b	d (con sólo un sujeto)		b
3ª	c (con sólo dos sujetos)	c	c (con sólo un sujeto)	b
4ª	b	b	c (con sólo un sujeto)	b
4ª (segunda prueba)	b	c (con sólo dos sujetos)	b	b
Nota adicional: Se ha probado correctamente la 4ª implementación con dos sujetos de cada tipo a la vez, excepto de tipo "Niño". Se asume que la prueba tiene éxito.				

Tabla 7: *Resultados de la cuarta batería de pruebas.*

Quinto hito

La batería de pruebas para el quinto hito es la siguiente:

1. Kinect con flujo esqueletal en Modo por Defecto. Módulo de salida configurado para producir los esqueletos de las personas. Sujeto de prueba de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. El sujeto mira en dirección al dispositivo Kinect de modo que se active el seguimiento facial, pero evita mirar fijamente a un punto. Se anotan los resultados:
    a. No se produce ninguna salida.
    b. Se produce correctamente un archivo de esqueleto con el tipo de persona que haya obtenido el módulo de reconocimiento. (No necesariamente el mismo que el tipo real de persona)
    c. Se produce correctamente un archivo de esqueleto con el tipo de persona reconocido, pero se crea en la ubicación por defecto en lugar de la asignada.
    d. Se produce un archivo con datos de esqueleto correctos, pero el tipo de persona reconocido es erróneo o no aparece.
    e. Se produce un archivo con datos de esqueleto erróneos (negativos o que no aparecen), pero con el tipo de persona reconocido correcto.
    f. Se produce un archivo con todos los datos erróneos.

2. Kinect con flujo esqueletal en Modo por Defecto. Módulo de salida configurado para producir los puntos de interés para las personas. Sujeto de prueba de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. El sujeto mira en dirección al dispositivo Kinect de modo que se active el seguimiento facial, y mira a un punto hasta que se reconozca una mirada fija. Se anotan los resultados:
    a. No se produce salida.
    b. Se produce correctamente un archivo de puntos de interés con el tipo de persona que haya obtenido el módulo de reconocimiento.
    c. Se produce correctamente un archivo de puntos de interés con el tipo de persona reconocido, pero se crea en la ubicación por defecto en lugar de la asignada.
    d. Se produce un archivo con datos de puntos de interés correctos, pero el tipo de persona reconocido es erróneo o no aparece.
    e. Se produce un archivo en el que no aparecen datos de puntos de interés, pero con el tipo de persona reconocido correcto.
    f. Se produce un archivo con todos los datos erróneos.

Una vez estas dos pruebas son correctas, se crea un archivo de puntos de interés para una persona siguiendo el procedimiento descrito más adelante y se analizan los resultados obtenidos. Se requiere un marco o algún tipo de indicador de la superficie del escaparate. Con esto, se pretende realizar dos pruebas adicionales para este hito que comprueban que todo se ha realizado correctamente.

3. Kinect con flujo esqueletal en Modo por Defecto. Módulo de salida configurado para producir los puntos de interés para las personas. Sujeto de prueba de pie de frente a la cámara, a una distancia suficiente para que se le vea el cuerpo completo. Marco colocado y sus coordenadas introducidas. Se pide al sujeto que mire a un punto concreto del marco, y se activa un cronómetro cuando el punto indicador de intersección llega a la zona. Una vez se ha convertido en el indicador de mirada fija, se pide al sujeto que deje de mirar y se para el cronómetro cuando lo haga. Se repite el proceso con un segundo punto. Se accede al archivo que se ha creado y se anotan los resultados:
    a. Los tiempos de mirada recogidos para ambos puntos en el archivo difieren de lo que se obtuvo al cronometrar al sujeto. (Aplicar medio segundo de tolerancia por errores humanos.)
    b. Los tiempos de mirada recogidos para ambos puntos en el archivo coinciden con lo que se obtuvo al cronometrar al sujeto.
    c. De los tiempos de mirada recogidos en el archivo, sólo el de uno de los puntos coincide con el que se obtuvo al cronometrar al sujeto.

4. Para el mismo archivo de puntos de interés, se usan las coordenadas indicadas en cada punto para medir la localización sobre el escaparate que se especifica que el sujeto estuvo mirando. Esta medición se realiza desde la esquina inferior izquierda del escaparate. Se anotan los resultados:
    a. Ninguna de las localizaciones de los puntos de interés coinciden con los puntos que se pidió al sujeto que mirara.
    b. Ambas localizaciones coinciden con los puntos que se pidió al sujeto que mirara.
    c. Sólo una de las localizaciones coincide con el punto específico que se pidió al sujeto que mirara.

Si en esta batería de pruebas no se obtiene un resultado de "b", significa que la prueba ha sido fallida y se debe revisar la implementación. Repetir el proceso de prueba, análisis del error y reparación del error hasta que se obtenga el resultado "b" en todas las pruebas. Los resultados de las pruebas realizadas se recogen en la Tabla 8.

Implementación	Result. Pr. 1	Result. Pr. 2		Result. Pr. 3	Result. Pr. 4
1ª	c	c			
2ª	b	b		b	b
Nota adicional: Se ha probado correctamente la 2ª implementación con dos sujetos.					

Tabla 8: *Resultados de la quinta batería de pruebas.*

Una vez se han superado las pruebas de este último hito, se da por finalizada la implementación del sistema. El prototipo es completamente funcional y sólo requiere de unas pruebas de validación para comprobar la plausibilidad de la prueba de concepto.

Pruebas de validación

Para la validación se prepararon una serie de pruebas, pero en un cierto momento se planteó la posibilidad de probar el sistema en el entorno para el que se había diseñado; realizar una prueba en el escaparate real de un comercio. Tras obtener las autorizaciones necesarias, se alteraron las pruebas de validación previstas para adaptarlas al nuevo escenario.

Los lugares de realización de las pruebas son dos comercios diferentes y en distintas horas del día. El primer comercio, un estanco denominado Expendeduría nº3 de Villalba, posee un escaparate externo encarado en dirección oeste, de modo que la luz solar es indirecta a lo largo del horario de mañana y pasa a ser directa durante la segunda mitad del horario de tarde. Sin embargo, la presencia de un techo y un toldo implica que la luz siempre será de espaldas al potencial cliente, nunca iluminará la parte delantera mientras se encuentre inspeccionando el escaparate.

El segundo comercio, Carnicería Fernanda, posee un mostrador interno y ventanas encaradas hacia el este, de modo que la luz solar es indirecta a lo largo de todo el día, pero de espaldas al potencial cliente. Además, en este último comercio se usa luz artificial durante toda la jornada, de modo que la iluminación del escenario es mixta.

Se realizarán varias sesiones de grabación en cada comercio. Se pretende estudiar los efectos del cambio de iluminación, de modo que el sistema se prueba en ambos comercios una vez en torno a mediodía y otra vez a principios del horario de tarde.

Para poder interpretar los resultados, se analizarán independientemente los puntos de interés que se han obtenido de los potenciales clientes y el tipo de persona que se ha reconocido de los mismos. El análisis de los puntos de interés será por comercio, mientras que el del tipo de persona reconocido será global, para así disponer de unos resultados generales de precisión.

Figura 13: *Vista trasera del escaparate del estanco.*

Expendeduría nº 3 de Villalba

El escaparate de este comercio es un escaparate estándar. Es plano, vertical, poco profundo y amplio, tanto horizontal como verticalmente. Dentro del escaparate hay todo tipo de artículos y varios estantes de cristal, a cuál más alto. La ubicación del dispositivo Kinect es una de estas estanterías de altura media, como se puede apreciar en las Figuras 13 y 14, que también dan una idea del aspecto del escaparate que se ha descrito.

Figura 14: *Vista delantera del escaparate del estanco.*

El entorno, como es normal, no está construido ni ubicado de la forma ideal para integrar un dispositivo Kinect. Por esta razón, se encontraron algunos problemas durante las pruebas, aunque se va a dejar de lado aquellos que no sean puramente prácticos, como la disponibilidad de enchufes.

El mayor inconveniente del escaparate es su poca profundidad combinada con su presencia en una acera muy transitada, lo que provoca que aquellos que van a mirar el escaparate se acerquen mucho al cristal. Esto hace que se encuentren demasiado cerca del dispositivo Kinect, con lo que su imagen se corta y no se puede activar el seguimiento facial. Un ejemplo de esto se puede apreciar en la Figura 15.

Otra desventaja del escaparate es la iluminación. El comercio cuenta con un techo sobre la zona de la entrada y el escaparate, extensible además por medio de un toldo. Esto hace que la claridad siempre venga de espaldas al cliente potencial, creando un halo en torno a su cuerpo, y especialmente en torno a su cabeza. Esto dificulta en buena medida la activación del seguimiento facial.

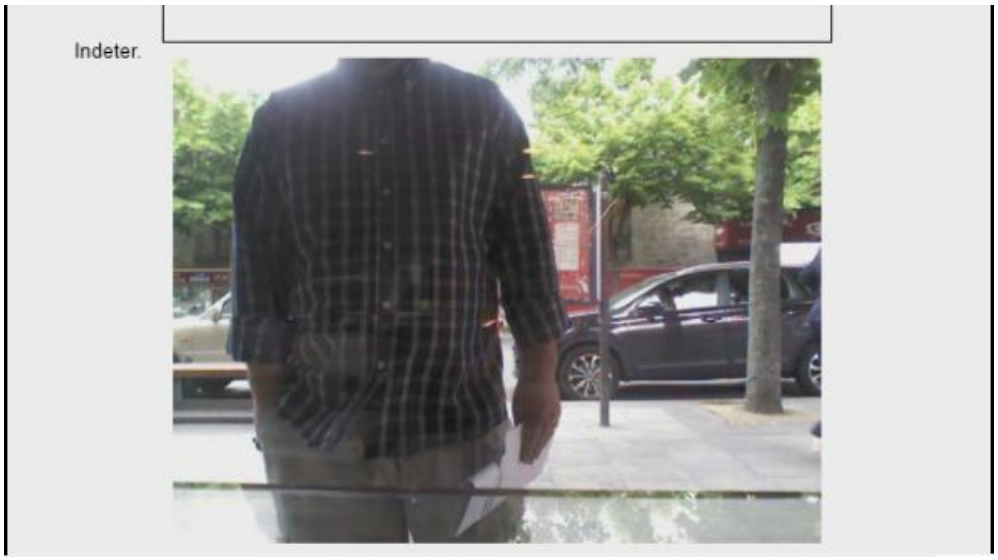


Figura 15: Ejemplo del inconveniente de la poca profundidad.

Para el análisis de los puntos de interés sobre el escaparate, se ha dividido la superficie de éste en nueve cuadrantes del mismo tamaño. De esta forma, se pueden distinguir las zonas a las que más se han mirado y se podría tratar de buscar una relación con respecto a lo que se ofrece. En el caso que nos ocupa, como indica la Figura 16, el cuadrante al que los potenciales clientes miraron fijamente más veces fue al inferior derecho, siendo la zona inferior la que más interesaba en general. Lo que se muestra en esa zona son libros, siendo el lado derecho el que posee más best-sellers que el resto de la zona inferior del escaparate, que muestra literatura más general.

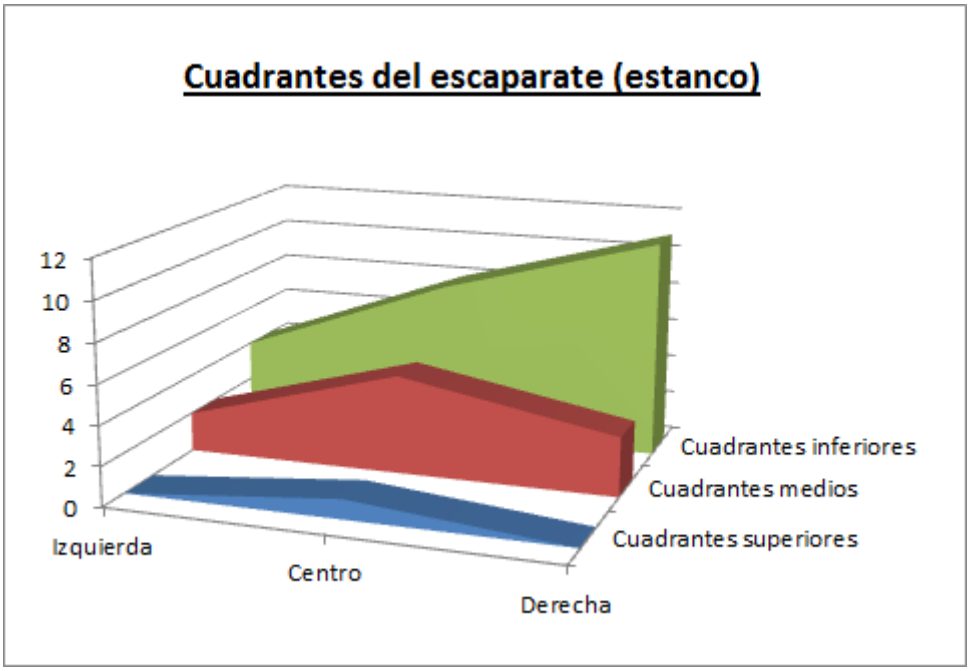


Figura 16: Gráfico de los puntos de interés por cuadrante en el estanco.

Figura 17: *Vista trasera del escaparate de la carnicería.*

Carnicería Fernanda

El escaparate de este comercio es un escaparate inclinado, ya que aprovecha la distribución del género en varias baldas escalonadas y permite a los clientes mirar más de cerca que si se tratara de un escaparate vertical. Por tanto, también cuenta con una buena profundidad que el dispositivo Kinect puede aprovechar mejor para captar el cuerpo de los clientes. Por otro lado, el escaparate completo tiene en realidad tres secciones, cada una de la cuales cambia ligeramente de dirección, lo cual hace imposible la supervisión de todas a la vez. La ubicación del dispositivo Kinect es detrás del escaparate sobre una encimera, en la sección principal del mismo, la que cuenta con la balanza. Esta zona se puede apreciar en las Figuras 17 y 18, que también dan una idea del aspecto del escaparate que se ha descrito, si bien no aparece el dispositivo Kinect.

Este entorno es más cercano al ideal para integrar el dispositivo Kinect al escaparate, ya que la distancia al cristal es óptima y permite ver una buena parte de las piernas de los clientes, a no ser que se peguen al escaparate. Sin embargo, en este caso el dispositivo Kinect no está centrado, y no se puede abarcar todo el escaparate. Por otro lado, también sigue afectando en buena medida la presencia de luz que viene de espaldas al cliente, dificultando el seguimiento facial. Una solución para ello es la presencia de una fuente de luz artificial sobre el escaparate, que ilumina la cara del sujeto y facilita la labor de seguimiento.

Figura 18: *Vista frontal del escaparate de la carnicería.*

Para el análisis de los puntos de interés sobre el escaparate, se ha dividido la superficie en nueve cuadrantes, del mismo modo que en el caso anterior. Aquí, como indica la Figura 19, los cuadrantes a los que los potenciales clientes más miraron con diferencia fueron los cuadrantes superior izquierdo y superior central. Esto se debe a que es la zona a la que llevan la carne que se ha pedido para cortarla y pesarla, de modo que es un lugar común al que prácticamente todos los clientes miran. Otra zona de interés puede ser la zona inferior/medio izquierda, que es la que mayor concentración de miradas tiene si se excluye la zona superior. Esto puede significar que ahí se expone la carne más vendida, o quizá la más deseada, cuanto menos.

General: reconocimiento de tipos

La aplicación obtiene automáticamente los datos del esqueleto de una persona a la que se ha conseguido realizar un seguimiento facial. Usando estos datos, se obtiene una clasificación por cada fotograma indicando a qué tipo de persona pertenece, "Hombre", "Mujer" o "Niño".

Debido a la programación interna del dispositivo Kinect, éste intentará siempre generar un esqueleto de una forma a la que haya identificado como una persona, incluso a pesar de que no cuente con una información completa. Debido a esto, es común observar un número de fallos en el esqueleto generado, sobre todo cuando alguna parte del cuerpo está ocluida por algún objeto, comúnmente brazos o piernas.

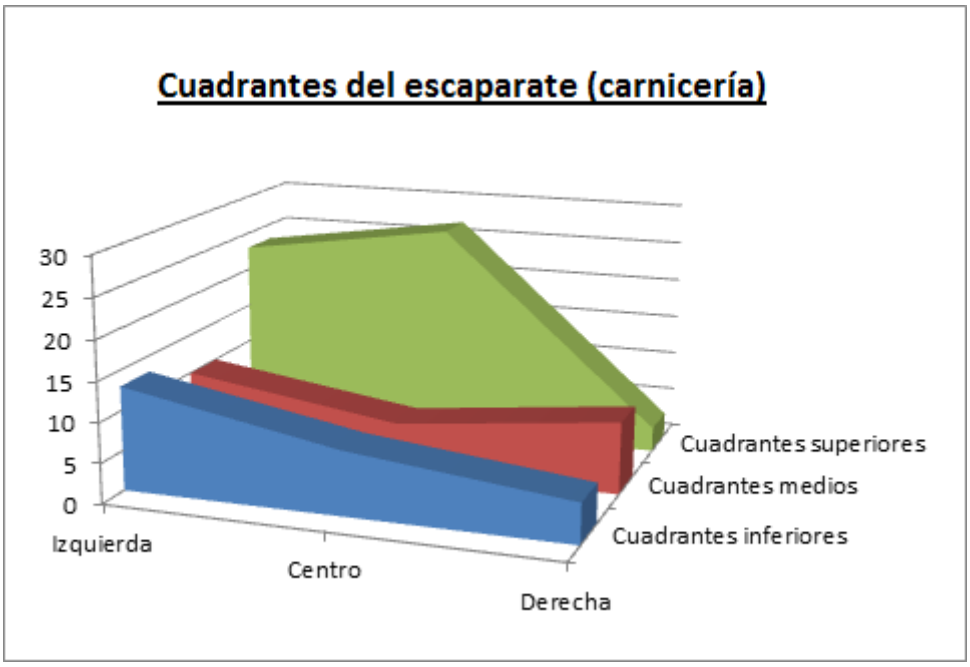


Figura 19: *Gráfica de los puntos de interés por cuadrante en la carnicería.*

Teniendo en cuenta estos fallos frecuentes en los datos esqueletales, que se ejemplifican en la Figura 20, se ha configurado al clasificador para que asigne un tipo a los datos de una persona cuando ésta haya abandonado la zona de visión del dispositivo, en lugar de la primera clasificación que se obtiene. Este tipo "definitivo" será simplemente el que más veces se haya obtenido durante el tiempo en que esa persona ha estado bajo seguimiento facial, otorgando así cierta robustez al clasificador.

Los datos obtenidos durante las pruebas se reflejan en la matriz de confusión que se muestra en la Tabla 9. En este tipo de matrices, la fila corresponde al elemento real y la columna corresponde al tipo "definitivo" en el que se le clasificó. Por tanto, como cabría esperar, unos números altos en las casillas que forman la diagonal de la tabla indican buenos resultados en la clasificación.

REAL \ RECONOCIDO	**Hombre**	**Mujer**	**Niño**	Total
Hombre	***6***	2	1	9
Mujer	0	***13***	9	22
Niño	0	0	***2***	2

Tabla 9: *Matriz de confusión con los tipos reales y los tipos reconocidos.*

Los resultados indican un reconocimiento exitoso en la mayoría de los casos. El número tan alto de clasificaciones de mujeres como niños se debe a que la mayoría de ellas se acercaron demasiado al escaparate de la carnicería y se perdió la información sobre sus piernas, obteniendo así esqueletos erróneos como el de la Figura 20.

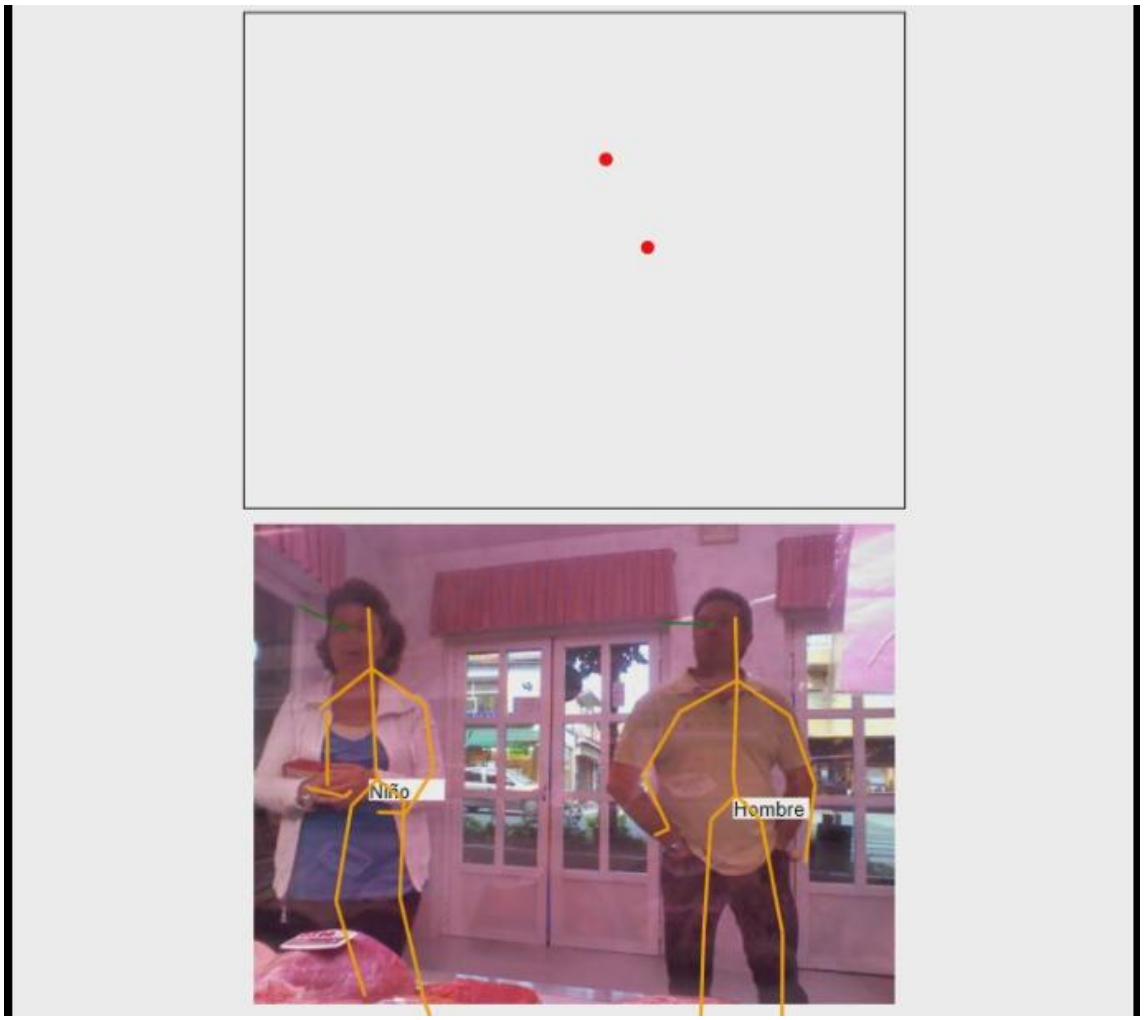


Figura 20: *Ejemplo de un reconocimiento de tipo exitoso y otro fallido debido a errores en el esqueleto por falta de información sobre las piernas.*

Conclusiones y futuros trabajos

A lo largo del desarrollo del proyecto se ha conseguido alcanzar un conjunto de objetivos que se detalla a continuación:

- Se ha realizado un exhaustivo estudio de las funcionalidades del nuevo SDK de Kinect para Windows y las aplicaciones adicionales que se encuentran en el Kit de Herramientas para Desarrolladores.
- Se ha analizado el código de ejemplo de una de las aplicaciones del Kit de Herramientas para Desarrolladores, mostrando su funcionalidad y las posibles partes que se pueden reutilizar en un futuro.
- Se ha desarrollado un sistema capaz de realizar un seguimiento facial de una persona con objeto de reconocer el lugar donde está mirando y cuál o cuáles de ellos despiertan su interés.
- Se ha implementado un sistema basado en redes de neuronas artificiales para determinar la edad y el género partiendo de la información esqueletal que el dispositivo Kinect suministra.
- Se ha diseñado e implementado un primer prototipo no comercializable del sistema de "escaparate caliente" que contabiliza los lugares de interés de un escaparate, haciendo una segmentación por sexo y edad.
- Se ha demostrado que la prueba de concepto ha superado las pruebas a las que ha sido sometida, suponiendo un rotundo éxito. No sólo se ha confirmado que un dispositivo Kinect puede ser integrado en un escaparate para obtener información sobre lo que llama la atención a los potenciales clientes, sino que también se ha demostrado que puede realizar sus funciones en un entorno alejado de lo idóneo.

El sistema de reconocimiento de tipos, también llamado de clasificación, merece una mención especial. Los resultados que se obtienen son inmensamente superiores a lo esperado, sobre todo si se tiene en cuenta que trabaja sobre los datos de otro sistema que en ocasiones fusiona a una persona con su carro de la compra, o con la persona de al lado, sin ir más lejos. Casi podría decirse que es un avance el poder reconocer el género de una persona con un coste computacional tan bajo para la precisión que se obtiene.

Futuros trabajos

El paso lógico una vez se conoce la viabilidad del "escaparate caliente" sería adaptar uno real para que funcione como tal. Un escaparate de un negocio en un centro comercial sería el lugar ideal, ya que evita que la luz provenga de detrás de las personas a la vez que cuenta, generalmente, con escaparates profundos donde el cristal llega hasta el suelo, permitiendo obtener toda la información de las piernas del potencial cliente. Un dispositivo Kinect perfectamente ubicado en dicho entorno, y con una aplicación completa en lugar de un prototipo, permitiría hacer uso de todo su potencial.

Una aplicación completa que se base en la prueba de concepto puede realizar muchas mejoras sobre su funcionamiento. Se puede adaptar para que no sólo reconozca escaparates planos y verticales, se puede mejorar la interfaz, se puede alterar para que no necesite los datos de un archivo de configuración y las salidas se guarden en una base de datos para que estén mejor organizadas. Se podría incluso modificar la base del prototipo y añadir de forma distribuida un dispositivo Kinect extra para que se pueda obtener más información o se consiga realizar un seguimiento a más gente a la vez.

Sin embargo, todas estas opciones se pueden considerar como secundarias. Lo que realmente podría considerarse una opción para futuros trabajos es la expansión y mejora del sistema de reconocimiento de tipos. Los resultados inesperados que este sistema obtuvo en las pruebas en entorno real hacen que esta opción resulte la más atractiva. Expandir este sistema para abarcar más tipos puede suponer un reto, pero sin duda merecerá la pena.

Las futuras aplicaciones de este sistema de reconocimiento de tipos pueden ser numerosas. Un ámbito especialmente indicado para el uso de este sistema es el del ocio, donde el reconocimiento preciso del sexo y la edad pueden resultar de gran ayuda. Una de las aplicaciones realistas en dicho ámbito en las que se puede utilizar este tipo de reconocimiento es un sistema de control parental, que restrinja o redirija a un menor a contenidos apropiados para su edad.

Bibliografía

[1] Microsoft Corporation. Kinect for Windows [en línea]: *Developer Downloads.* <http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx> [Consulta: 25 de junio de 2013]

[2] Microsoft Corporation. Microsoft Developer Network [en línea]: *Developer Toolkit version 1.5.0.* <http://msdn.microsoft.com/en-us/library/jj663803.aspx#SDK_1_5__DTK_1_5_0> [Consulta: 13 de junio de 2013]

[3] Microsoft Corporation. Microsoft Developer Network [en línea]: *System requirements.* <http://msdn.microsoft.com/en-us/library/hh855359.aspx> [Consulta: 13 de junio de 2013]

[4] Microsoft Corporation. Microsoft Developer Network [en línea]: *Face Tracking.* <http://msdn.microsoft.com/en-us/library/jj130970.aspx> [Consulta: 15 de junio de 2013]

[5] M. Rydfalk. CANDIDE, a parameterized face, Informe Nº LiTH-ISY-I-866, Departamento de Ingeniería Eléctrica, Universidad de Linköping, Suecia, 1987.

[6] Jörgen Ahlberg. CANDIDE - a parameterized face [en línea]: *The CANDIDE versions*. Universidad de Linköping. <http://www.icg.isy.liu.se/candide/> [Consulta: 15 de junio de 2013]

[7] Cámara oficial de Comercio, Industria y Navegación de Valencia. *Conceptos básicos de escaparatismo* [en línea]. Sin información sobre edición, actualizado en septiembre de 2011. Disponible en web: <http://www.camaravalencia.com/es-ES/servicios/comercio/informes_publicaciones_comercio/Documents/CUADERNOS%20COMERCIO%202011/Camara-FolletoEscaparatismo2011-baja.pdf>

Anexo 1: Presupuesto

La aplicación desarrollada durante el transcurso del presente proyecto se ha pensado como prueba de concepto, no como producto final. Por esa razón, no tiene sentido realizar un presupuesto como tal, ya que al final del proceso no se va a contar con una aplicación susceptible de ser vendida.

Así pues, lo que se va a realizar durante este apartado será una valoración de gastos en lugar de un presupuesto. Con ello, se pretende reflejar los costes asociados al tiempo invertido en el desarrollo de la prueba de concepto, de modo que la cifra final será mucho menor que la de un presupuesto para una aplicación con un tiempo de desarrollo similar, al no incluir impuestos ni beneficios.

Gastos

Los gastos derivados del proceso de creación de la prueba de concepto, que se desarrolló durante tres meses y medio, ascienden a un total de 5.271,73 euros. Las fuentes de dichos gastos se desglosan a continuación.

Desglose de costes directos

PERSONAL

Apellidos y nombre	Categoría	Dedicación en horas de trabajo	Coste por hora	Coste (Euros)
Sibón Sancho, Jose Enrique	Jefe de proyecto	32	25,00	800,00
Sibón Sancho, Jose Enrique	Analista	92	20,00	1.840,00
Sibón Sancho, Jose Enrique	Diseñador	60	13,00	780,00
Sibón Sancho, Jose Enrique	Programador	80	9,00	720,00
Sibón Sancho, Jose Enrique	Tester	28	4,00	112,00
Sibón Sancho, Jose Enrique	Redactor	64	11,00	704,00
			Total	4.956,00

EQUIPOS

Descripción	Coste (Euros)	% de uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable[b)]
Estación de trabajo	0,00[a)]	100	3,2	60	0,00
Dispositivo Kinect	0,00[a)]	100	0,5	60	0,00
Toshiba Qosmio	1.149,99	0,03	1	60	19,17
				Total	19,17

[a)]Artículo proporcionado y financiado por la Universidad Carlos III de Madrid. No se posee la propiedad de este artículo, sólo permiso para su uso. En caso de software, se proporciona una licencia gratuita por un año.

[b)]Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses de uso del equipo desde su facturación.
B = periodo de depreciación.
C = coste del equipo.
D = porcentaje de uso dedicado al proyecto.

OTROS COSTES DIRECTOS DEL PROYECTO[c)]

Descripción	Empresa	Coste imputable
Internet	Telefónica	195,84
Electricidad	Iberdrola	47,52
Viajes	-	53,20
Microsoft Office 2010	Microsoft	0,00[a)]
Microsoft Visual Studio 2010	Microsoft	0,00[a)]
	Total	296,56

[c)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas.

Resumen de gastos

Concepto	Costes totales
Personal	4.956,00
Equipos	19,17
Costes de funcionamiento	295,56
Total	**5.270,73**

Anexo 2: Planificación y diagrama de Gantt

La planificación inicial del proyecto se detalla en la Figura 21 mostrada a continuación. En ella, se puede apreciar que el proyecto tiene lugar a lo largo de tres meses y medio de intenso trabajo, distribuidos en varios ciclos en los que se persiguen objetivos específicos. En el diagrama de Gantt que se encuentra ilustrado en las Figuras 22 y 23 de la siguiente página se puede observar la duración relativa de cada una de las tareas.

Nombre	Fecha de inicio	Fecha de fin
Proceso de documentación preliminar	11/03/13	14/03/13
Proceso de documentación sobre Kinect y su SDK	13/03/13	21/03/13
Proceso de documentación sobre seguimiento facial	20/03/13	21/03/13
Análisis del código de ejemplo	22/03/13	1/04/13
Primer ciclo: modificación del ejemplo	2/04/13	10/04/13
Etapa de inicialización 1	2/04/13	8/04/13
Hito: Obtener vector de mirada	9/04/13	9/04/13
Etapa de iteración 1	9/04/13	10/04/13
Segundo ciclo: módulos de escaparate y cálculo	11/04/13	19/04/13
Etapa de inicialización 2	11/04/13	17/04/13
Hito: Obtener intersección con escaparate	18/04/13	18/04/13
Etapa de iteración 2	18/04/13	19/04/13
Tercer ciclo: escaparate - obtención de mirada fija	22/04/13	25/04/13
Etapa de inicialización 3	22/04/13	23/04/13
Hito: Obtener indicador de mirada fija	24/04/13	24/04/13
Etapa de iteración 3	24/04/13	25/04/13
Cuarto ciclo: módulo de clasificación	26/04/13	6/05/13
Etapa de inicialización 4	26/04/13	1/05/13
Hito: distinción por tipos de persona	2/05/13	2/05/13
Etapa de iteración 4	2/05/13	6/05/13
Quinto ciclo: módulos de reunión de datos e I/O	7/05/13	13/05/13
Etapa de inicialización 5	7/05/13	9/05/13
Hito: registrar todos los datos de una sesión	10/05/13	10/05/13
Etapa de iteración 5	10/05/13	13/05/13
Pruebas de validación	15/05/13	17/05/13
Redacción de la memoria	14/05/13	24/06/13

Figura 21: *Planificación inicial del proyecto.*

Se considera que no es necesario elaborar una nueva planificación ni un nuevo diagrama de Gantt con los datos reales de tiempo debido a la pequeña diferencia que existe con respecto a lo planeado. En su lugar, se detallan los cambios a continuación:

- El Proceso de documentación sobre Kinect y su SDK comienza un día más tarde, pero su duración también se reduce en la misma medida.
- En el Análisis del código de ejemplo se tarda un día menos de lo previsto.
- En el Segundo ciclo, se alcanza el hito tres días antes de lo planeado, reduciendo en dos días la duración de la Etapa de inicialización 2.
- En el Cuarto ciclo, se tarda un día más en la Etapa de inicialización 4 y otro día más en empezar la Etapa de iteración 4. Sin embargo, gracias al adelanto acumulado, no se observa un retraso sobre la planificación.
- Las Pruebas de validación comienzan un día más tarde de lo previsto.

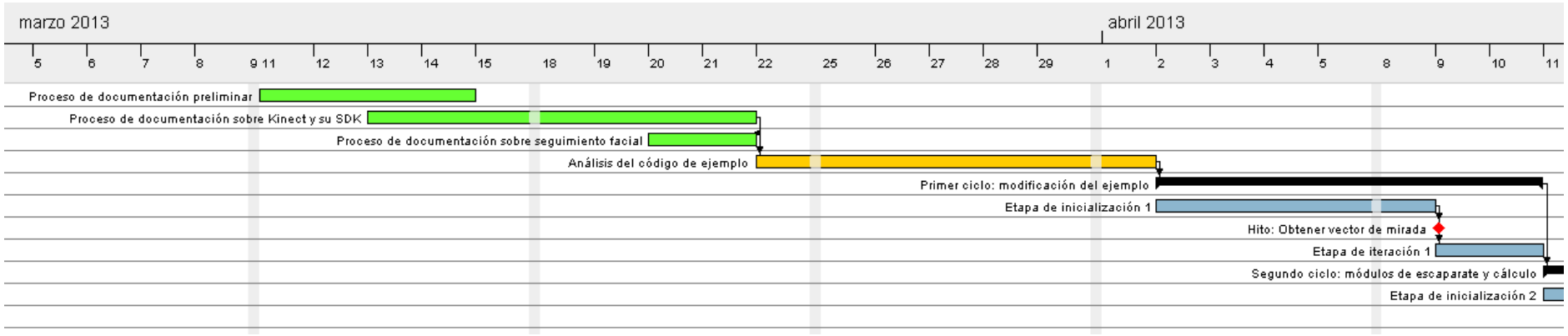


Figura 22: *Primera mitad del diagrama de Gantt de la planificación.*

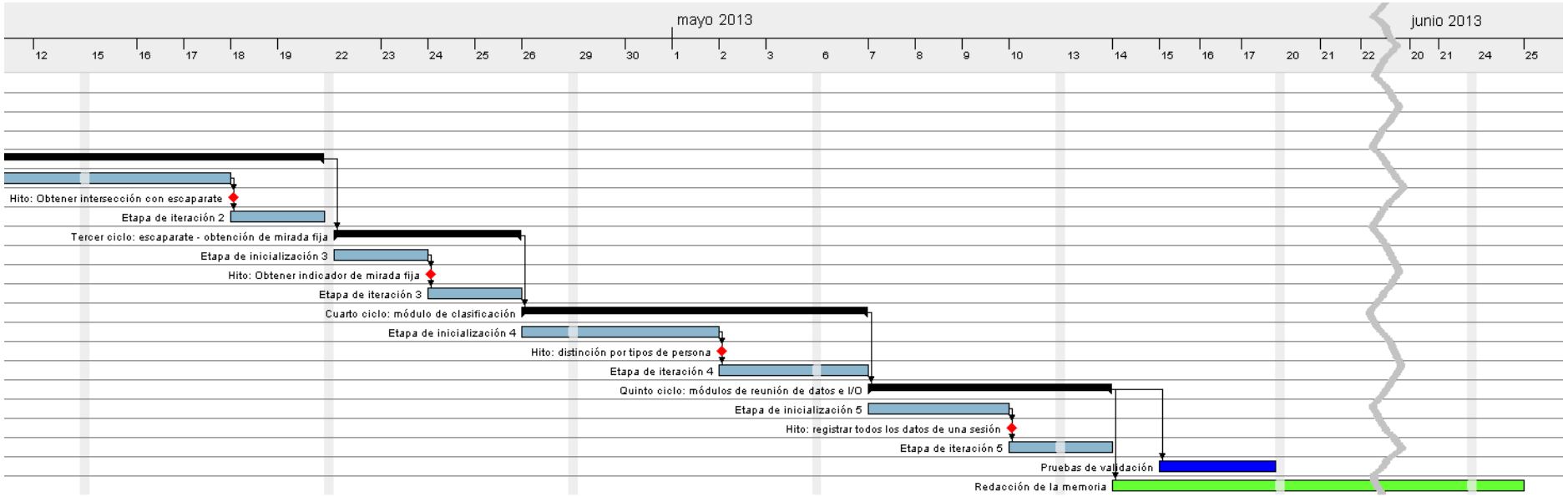


Figura 23: *Segunda mitad del diagrama de Gantt de la planificación.*